1990-03

Coupling artificial intelligence and a system dynamics simulation to optimize quality assurance and testing in software development

Agan, Christopher E.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

COUPLING ARTIFICIAL INTELLIGENCE AND
A SYSTEM DYNAMICS SIMULATION TO
OPTIMIZE QUALITY ASSURANCE AND TESTING
IN SOFTWARE DEVELOPMENT

by

Christopher E. Agan

March, 1990

Thesis Advisor: Tarek K. Abdel-Hamid

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 55	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No / Project No / Task No / Work Unit Accession Number

11. TITLE (*Include Security Classification*)

COUPLING ARTIFICIAL INTELLIGENCE AND A SYSTEM DYNAMICS SIMULATION TO OPTIMIZE QUALITY ASSURANCE AND TESTING IN SOFTWARE DEVELOPMENT

12. PERSONAL AUTHOR(S) Agan, Christopher E.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	March 1990	166

16. SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	Artificial Intelligence, Expert Systems, Project Management, Quality Assurance, Testing, Simulation

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

The allocation of effort to quality assurance and testing is vitally important to the successful development and maintenance of a software system. There is no quantitative method for finding the right allocation policy. The most common methods include allocating a fixed percentage of effort for all software projects or using allocations that have been used for similar projects in the past. The benefits of choosing the correct manpower allocation to suit a particular project can be substantial.

Using the System Dynamics Model of Software Project Management an optimal quality assurance and testing level for a project's development lifecycle can be found. The focus of this thesis is to design an expert system that can be coupled with the model in order to find the optimal allocation of quality assurance and testing effort for a particular project.

Two expert system modules were developed, that when coupled with the system dynamics model, will find the optimum quality assurance and testing distributions for a software project. The expert system modules were then used to perform sensitivity analysis experiments on the results.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Tarek K. Abdel-Hamid	22b. TELEPHONE (*Include Area code*) (408) 646-2686 / 22c. OFFICE SYMBOL 54AH

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

Coupling Artificial Intelligence and
a System Dynamics Simulation to
Optimize Quality Assurance and Testing
in Software Development.

by

Christopher E. Agan
Lieutenant, United States Navy
B.S., University of Rochester, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March, 1990

Author: [redacted]
Christopher E. Agan

Approved by: [redacted]
Tarek K. Abdel-Hamid, Thesis Advisor

[redacted]
Tung Bui, Second Reader

[redacted]
David R. Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

The allocation of effort to quality assurance and testing is vitally important to the successful development and maintenance of a software system. There is no quantitative method for finding the right allocation policy. The most common methods include allocating a fixed percentage of effort for all software projects or using allocations that have been used for similar projects in the past. The benefits of choosing the correct manpower allocation to suit a particular project can be substantial.

Using the System Dynamics Model of Software Project Management an optimal quality assurance and testing level for a project's development lifecycle can be found. The focus of this thesis is to design an expert system that can be coupled with the model in order to find the optimal allocation of quality assurance and testing effort for a particular project.

Two expert system modules were developed, that when coupled with the system dynamics model, will find the optimum quality assurance and testing distributions for a software project. The expert system modules were then used to perform sensitivity analysis experiments on the results.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

In recent years the cost of computer hardware has decreased while the performance has increased significantly. Meanwhile, the demand for more sophisticated software continues to grow as computers become a more important part of everyday life. The development of software continues to be plagued by cost overruns and schedule delays. While a number of methodologies have been developed to address the technical problems associated with software development few solutions to the managerial problems have been presented [Ref. 1: p. 2].

A computer based model of the software development process can be used to study the effects of altering certain management controlled variables on the cost and completion time of a project. Coupled with an expert system, the model can simulate the same project many times with different parameters in order to find the values that result in the lowest cost.

B. OBJECTIVES

A comprehensive systems dynamics model of the software development process has been developed that can serve as an experimentation vehicle for studying the effects altering certain management policies [Ref. 1: p. 3]. Currently a naive expert system module exists that will generate an optimal level and distribution of quality assurance effort for a software development process. However, this system does not take into account the effect that altering quality assurance (QA) policy will have on other inputs such as the allocation of effort for testing.

Quality assurance methods such as reviews, walkthroughs, code reading, etc., are designed to detect errors as early as possible in the software development lifecycle.

> In a study by Shooman reported in McClure (1981), it was determined that detecting and correcting a design error during the design phase (i. e., through QA activities) is one-tenth the effort that would be needed to detect and correct it later during the systems testing phase because of this additional inventory of specifications, code, user and maintenance manuals, ... etc, that would require correction in the later case [Ref. 2: p. 217].

It follows from this that any change in the QA policy for a project will also effect the effort required in the testing phase. To determine the extent of this change the expert system would need to make adjustments to the effort allocated to testing as well as QA to find the point of optimality for both. The purpose of this thesis is to design and implement an expert system module that will allow for the simultaneous optimization of two variables. The test case presented will be that of quality assurance and testing however, the module is designed to be generic, so that any two interrelated variables may be optimized.

C. SEARCH STRATEGIES

The process of finding a goal state through the application of various operators is defined in artificial intelligence terms as search [Ref. 3]. Optimization of two or more variables in an expert module involves the search for the optimal solution by applying the operators to adjust the level of a particular variable. Various AI search techniques have been developed, from simpler ones that visit each state in order to find the goal state to strategies that use evaluation and cost functions to find the best path to a goal. Figure 1-1 summarizes some of the classic AI search methods [Ref. 3: p. 197].

Before describing search strategies some definitions must be presented [Ref. 3: pp. 192-202].

- A *state* is a situation in the world described by some facts. In the case of the expert system a state is the current values for the two variables and the project cost after executing the model with the those values.
- An *operator* is an action that when taken results in moving from one state to another. An example is the adjustment up or down of the values of the variables being optimized by the expert system.
- *Heuristics* are any nonnumeric advice about what order to apply operators to move from one state to another. An example might be always adjust QA before testing because QA effects testing.
- An *evaluation function* provides a means for ranking states numerically so that the state with the best number is chosen next. By convention evaluation functions are nonnegative where the smaller the number the better the state, with the goal state having evaluation function zero.
- A *cost function* is a nonnegative number measuring the cost associated with going from one state to another.
- An *agenda* is a list of states that have been found, but whose successor states have not yet been found.

1. Simple Search Strategies

The simplest search strategies to understand and implement are depth-first and breadth-first search. In both cases neither cost or evaluation functions are used. Heuristics are often used to help choose which of the successor states will be tried next.

With depth-first search, first the start state is visited and then some successor to that state, and then a successor of that, until the goal state is reached or a state without successors is reached [Ref. 3: p. 197]. If the latter situation is encountered the search returns to the immediate predecessor and continues with another successor. This pattern continues until the goal is found or no more states are left unvisited.

Breadth-first search first visits the start state, then each of its successors and then each of their successors until the goal state is found or no more states are left unvisited [Ref. 3: p. 198]. Breadth-first search is guaranteed to find the goal state if one exits.

Name of search stategy	*Uses agenda?*	*Uses evaluation function*	*Uses cost function*	*Next state whose successors are found*
Depth-first search	no	no	no	A successor of the last state, else a successor of a predecessor
Breadth-first search	yes	no	no	The state on the agenda the longest
Hill-climbing	no	yes	no	The lowest-evaluation successor of the last state
Best-first search	yes	yes	no	The state on the agenda of lowest evaluation function
Branch and bound	yes	no	yes	The state on the agenda of lowest total cost
A* search	yes	yes	yes	The state on the agenda of lowest sum of evaluation value and total cost

Figure 1-1. Classic Search Strategies

2. Searches Using Evaluation Functions

The evaluation function variant of depth-first search is called hill-climbing. In this case instead of arbitrarily choosing (or using a heuristic) which successor state to visit next, the state with the lowest evaluation function is visited first [Ref. 3: p. 201].

Breadth-first search also has an evaluation function variant called best-first search. This method does however, include a slight variation from breadth-first search. Instead of choosing the state with the lowest evaluation function on the same level, an agenda

is used and the state on the agenda with the lowest evaluation function is chosen. This state need not be on the same level as the previous state [Ref. 3: p. 201].

3. Optimal Path Search

Searches under this category are used when instead of just any path, the lowest cost path to the goal need be found. If a cost function can be found but no good evaluation function can be found then branch-and-bound search can be used. This strategy is similar to best-first search except that the state with the lowest cost, not evaluation function, is chosen from the agenda [Ref. 3: p. 203].

When both a cost and evaluation function can be found then A* can be used. In this strategy, the cost and evaluation functions are summed together and this number is used, like the evaluation function in best-first search, to choose the lowest number from the agenda [Ref. 3: p. 203]. The first path found to the goal using A* search is guaranteed to be the lowest cost path.

II. CURRENT SYSTEM CONFIGURATION

A. A SYSTEM DYNAMICS MODEL OF SOFTWARE DEVELOPMENT

As part of a wide-ranging study of software project management, a comprehensive system dynamics model of the software development process has been developed. The model serves as a "laboratory tool" for conducting experimentation into the dynamics of software development [Ref. 1:p. 7]. Coupling the model with an expert system creates a tool that can make changes to input variables and feed them into the model for testing and use feedback from the model to determine if further adjustment is required.

The System Dynamics Simulation model was developed on the basis of field interviews with software project managers in five organizations, complemented by an extensive database of empirical findings from the literature [Ref. 4]. The model is written in Professional Dynamo, a continuous simulation language developed in the late 1950's at M. I. T. It can perform several important functions, including its main goal of aiding the project manager in understanding the software development process. The manager can use the model as a decision aid by performing "what-if" experiments to develop a more complete understanding of the interrelationships of software development variables [Ref. 5:p. 8].

The model can also be used to help the project manager in the management of an actual software project. For example, the model can be used to estimate the total project cost and completion time. Input variables such as Fraction of Manpower Devoted to Quality Assurance (TPFMQA) or Willingness to Change Workforce (WCWF) can be changed and the simulation run to get feedback on the result of the

change. This allows the manager to make decisions on different managerial strategies for completing the project. [Ref. 1:pp. 7-8].

The model combines the multiple functions of the software development process. It includes both management-type function (e.g., planning, controlling, staffing) and software production-type activities (e.g., design, coding, reviewing, testing). This integrative technique is effective in identifying the multiple factors that are compounded to cause software project problems [Ref 1: p. 7].

Another characteristic of the model is the use of system dynamics feedback principles to organize and explain the complex structure of dynamically interacting variables involved in the development and management of software projects. Feedback is the mechanism in which an action taken by an entity will ultimately effect that entity [Ref. 1:p. 7].

A third feature of the system dynamics model is its use of the computer simulation tools of system dynamics to deal with the highly complex integrative feedback model. Even though the dynamic effects of single feedback loops may be quite obvious, the actions of systems with interconnected feedback loops will often confuse human intuition. Because of the complexity of the feedback structures present in many real problems, a problem's behavior over time may only be traceable through the use of simulation techniques [Ref. 1:pp. 7-8]. Figure 2-1 is an overview of the simulation model's four major subsystems: (1) the human resource management subsystem; (2) the software production subsystem; (3) the controlling subsystem; and (4) the planning subsystem [Ref. 1:p. 9]. The remainder of this section provides a brief overview of the four subsystems with emphasis on the quality assurance and testing components[1].

---

[1] The description of the four major subsystems is taken directly from Abdel-Hamid, T. K., and Madnick, S. E., *Software Project Management*, pp. 115-117.

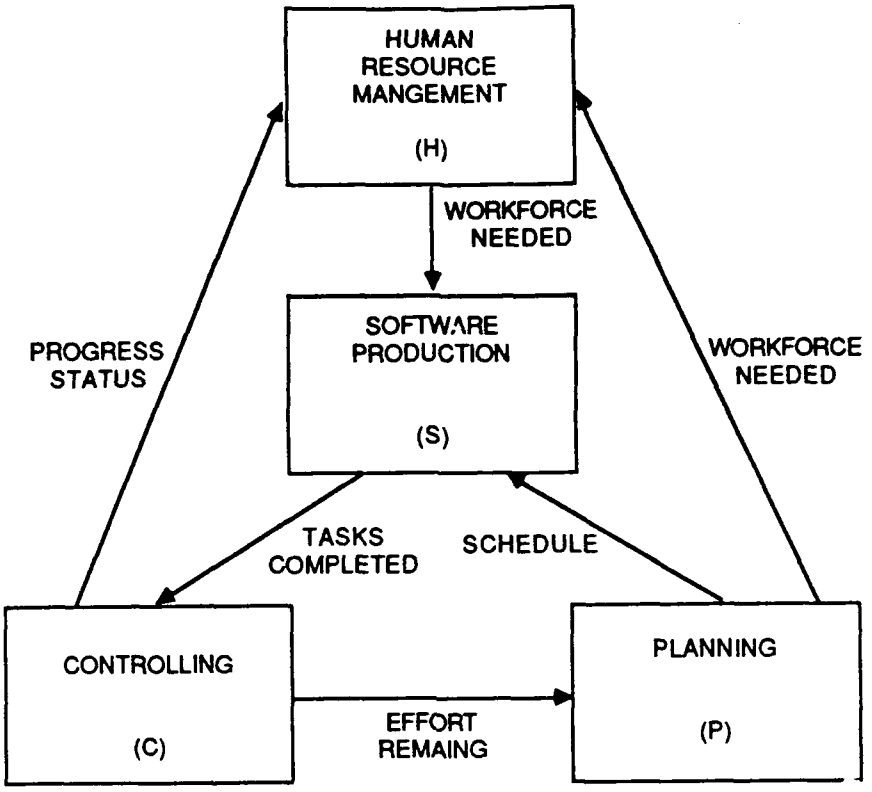

Figure 2-1. Four Subsystems of System Dynamics Simulator

The Human Resource Management Subsystem captures the hiring, training, assimilation and transfer of a project's human resources. Such actions are not carried out in a vacuum, they, as Figure 2-1 suggests, both affect and are effected by the other subsystems. For example, the project's "hiring rate" is a function of the "workforce needed" to complete the project on a planned completion date.

Similarly, the "workforce available", has a direct bearing on the allocation of manpower among the different software production activities in the Software Production Subsystem. The four primary software production activities are development, quality assurance, rework, and testing. The development activity

comprises both the design and coding of software. As the software is developed it is also reviewed e.g., using structured-walkthroughs, to detect any design/coding errors. Errors detected through quality assurance activities are then reworked. Not all errors will be detected and reworked, however, some will "escape" detection until the testing phase.

As progress is made, it is reported. A comparison of where the project is versus where it should be (according to plan) is a control-type activity captured within the Controlling Subsystem. Once an assessment of the project's status is made, it becomes an important input to the planning function.

In the Planning Subsystem, the initial project estimates are made to start the project, and then those estimates are revised, when necessary, throughout the project's life. For example, to handle a project that is perceived to be behind schedule, plans can be revised to (among other things) hire more people, extend the schedule, or do a little of both.

1. Quality Assurance Component

The quality assurance component is part of the Software Production Subsystem. A primary objective of quality assurance (QA) activities is the detection and correction of software errors that have been generated. These activities typically consist of pre-planned group meetings to review all tasks developed since the previous review. Since the objective of these activities is to detect errors and undetected errors are by nature invisible it is difficult to tell how adequate the QA job was until the testing phase is completed [Ref. 4:p. 6].

The allocation of manpower to quality assurance activities is handled by the manpower allocation component of the Software Development Subsystem. In the model the variable "Planned Fraction of Manpower for QA" (TPFMQA) is used to determine the "Actual Fraction of Manpower for QA" (AFMPQA). These two values

may be different due to schedule pressures since as schedule pressures mount, QA activities are often relaxed [Ref. 2:p. 138]. The variable TPFMQA is a "table function" that depicts a nonlinear relationship between percent of development phase completed and the percentage of manpower to be allocated to QA. This value can be changed by the user to experiment with different QA policies. Figure 2-2 shows the planned QA distribution used by NASA on the DE-A software project.

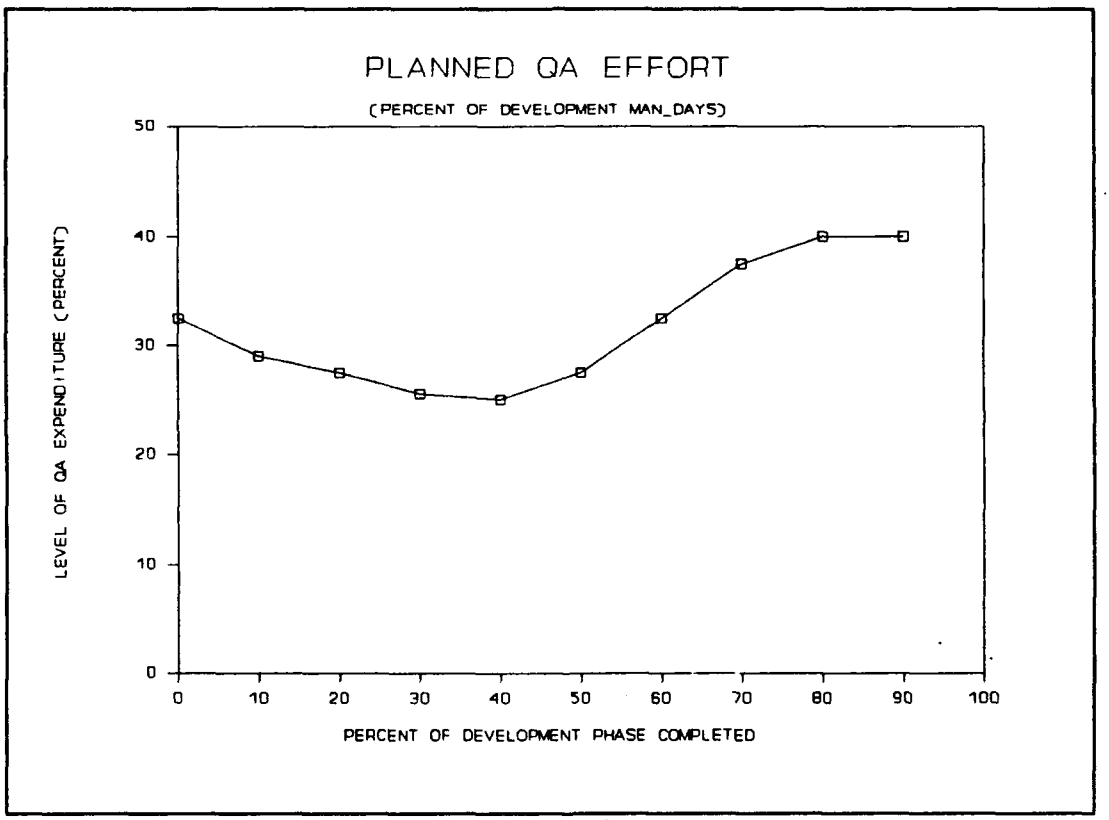

Figure 2-2.
QA Distribution for NASA's DE-A Project

2. System Testing Component

The testing component is also part of the Software Production Subsystem. The purpose of system testing is the detection and correction of errors that were not

detected by QA activities as well as those caused by bad fixes that result from faulty rework [Ref. 2:p. 215].

The variable "Percent of Effort Assumed Needed for Development" is used by the Controlling Subsystem to estimate the total man-days that will be required for both testing and development. First the total project effort is estimated (e.g.,using an estimation model such as COCOMO). This estimate is then multiplied by DEVPRT to estimate the man-days required for development, and by (1 - DEVPRT) to estimate the man-days required for testing. For example if the project cost is estimated to be 2100 man-days and DEVPRT is set to 0.85 then the estimate of the effort for development will be:

$$0.85 * 2100 = 1785 \text{ man-days.}$$

Simliarly the effort for testing will be estimated as:

$$(1 - 0.85) * 2100 = 315 \text{ man-days.}$$

The estimate of testing effort along with the perceived project size in tasks is used to determine the initial estimate for testing productivity. For example, if the project size is estimated at 100 tasks and the testing effort is estimated at 20 man-days then the initial estimate for testing productivity is 5 tasks/man-day [Ref. 2:p. 244].

B. QUALITY ASSURANCE ALLOCATION MODULE

An expert module to determine the optimal allocation of quality assurance effort for a software development project was developed, as part of a previous research effort, to interface with the simulation module [Ref 5]. This module was designed to derive QA schemes, execute the model to test them and use the feedback from the results to improve the efficiency of the QA distribution. A naive algorithm of applying positive and negative pulses to each of the points in the TPFMQA table was used to accomplish the optimization. The basics of the algorithm are as follows:

1. Start with a QA distribution input by the user. At each point in the table first apply a negative pulse[2] and execute the model to determine the result. For example if the user entered a 15% uniform QA distribution to start and a pulse size of 0.15 then the new first value for TPFMQA would be: 0.15 - (0.15 * 0.15) = 0.1275.

2. If the negative pulse does not provide improvement then apply a positive pulse and again execute the model to test the result. Using the same staring values as the above example the new first value for TPFMQA would be: 0.15 + (0.15 * 0.15) = 0.1725.

3. If neither pulse yields improvement then return the variable to its original value.

4. Steps 1-3 are performed for each of the ten points in the TPFMQA table. Once the cycle is complete, the procedure repeats again from the first point.

5. The algorithm terminates when either: (1) the percent improvement in total cost from one cycle to the next is less than the user specified exit condition; or (2) the maximum number of cycles is exceeded.

The next chapter describes the design of a new expert module that can be used to allocate QA effort. A different algorithm is used that decreases the number of cycles that is required to find the optimal point.

[2]The user enters a pulse size which is a percentage of the current value. The current value is then either increased or decreased by this percentage to create the new value for testing with the model.

III. EXPERT SYSTEM MODULES

A. INTRODUCTION

This chapter will provide a description of the four expert systems modules, namely: (1) single.ari; (2) table.ari; (3) depth.ari; and (4) breadth.ari; that were designed and implemented as part of this thesis. A description of the algorithms and the important rules will be given for each. Additionally instructions for use of each program will be given. A complete listing of the documented source code for each program is given in Appendices A-D.

B. PROGRAM REQUIREMENTS

Each of the programs described in this chapter were written for use on an IBM PC or compatible running the Arity Prolog Interpreter. Additionally, each program is designed to interface with the System Dynamics Model for Software Project Management and as such will require the following files be located in the same directory as the program when it is executing:

1. PROJECT.DYN - This is the file containing the Dynamo code for the simulation model that is discussed in Chapter II.

2. PROJECT.DRS - This file is used after a simulation run by the Dynamo Report Generator to write the value for CUMMD (the Cumulative Man-Days for the simulated project) to the file PROJECT.OUT in the form of a prolog predicate so it can be read by the expert system.

3. MODEL.BAT - This is a batch file containing the necessary DOS commands to simulate a project and to store the resulting value for CUMMD into the file PROJECT.OUT. This program executes the following Professional Dynamo files that must also be in the default directory: (1) DYNEX.EXE; (2) SMLT.EXE; and (3) REP.EXE.

Figure 3-1 shows the major components of the expert system simulation model architecture and the data flows between them.

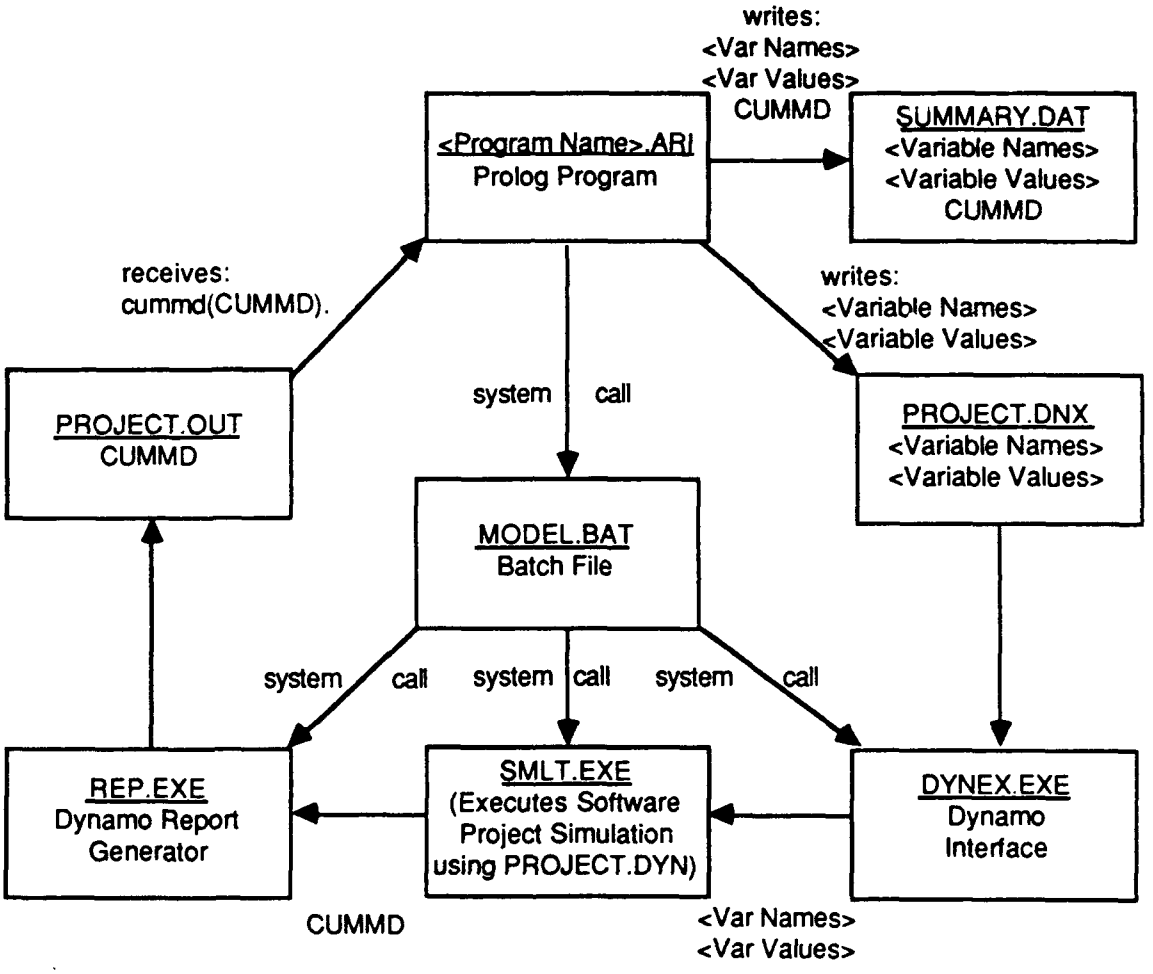

Figure 3-1. The Expert System Simulation Model Architecture

C. EXPERT MODULE SINGLE.ARI

The first step in implementing a module that can optimize two different variables was to design and implement an expert system module that could optimize a single

valued constant variable in the model. An example of this type of variable is the "Percent of Effort Assumed Needed for Development" (DEVPRT). The value for DEVPRT is initialized in the model to 0.85, however, this value can be adjusted to test the result of different effort allocations on total project cost as measured by the variable CUMMD (Cumulative Man-Days). The purpose of this module is to find the value for a user supplied variable that results in the lowest project cost with all other variables held constant.

1. Algorithm Description

The general idea behind this algorithm is to first find the direction of improvement from the user supplied starting point by adding and subtracting a static pulse from the initial value and comparing the resulting CUMMD with the value found at the starting point (e.g., subtract pulse 0.05 from starting value DEVPRT = 0.85 and find the value for CUMMD with DEVPRT = 0.8). Whichever pulse direction, positive or negative, that yields an improved value for CUMMD becomes the current direction of improvement. Movement continues in this direction until improvement is no longer found, at which point the pulse size is reduced and the procedure starts over. This pattern continues until an exit condition is met. The entire process is then repeated from a second starting point to see if the same answer is found. This technique of starting from two different points is used to overcome the problem of a bi-modal distribution. A detailed description of the algorithm follows:

1. Query the user for the initial values for the name of the variable to be tested, the pulse size[3], the minimum pulse size (to be used to terminate the algorithm), and two starting points.

2. Starting at the first point find the value for "Cumulative Man-Days Expended" (CUMMD) at that point by executing the model.

---

[3]For the four applications described in this thesis, the pulse size is an absolute amount that is added to or subtracted from the current value to get the new value for the variable.

3. Test if the exit condition has been met. The algorithm is terminated when the value for the pulse size falls below the user supplied minimum value.

4. Find the direction of improvement from the starting point. This is accomplished by first subtracting the value of the pulse from the current value for the variable and then executing the model to find the new value CUMMD. If improvement is found then no further action is taken and the algorithm proceeds with the next step. If improvement is not noted, first the value is returned to the starting point and then a positive pulse is added to the variable and the result is tested for improvement. Again if improvment is found the algorithm proceeds from the next point. If neither pulse results in improvement then the variable is returned to its original value, the size of the pulse is halved, and the algorithm returns to step 3.

5. If a direction of improvement is found then continue moving in that direction, one pulse width at a time, until improvement is no longer noted.

6. Halve the pulse size and return to step 3. This search starts from the point that resulted in the minimum value for the previous cycle.

7. Once the exit condition is met and the minimum value from the first starting point is found, start the algorithm over using the second starting point supplied by the user. If the two results are not the same then the value that results in the lowest cost is chosen as the true minimum.

2. Example Using the Algorithm

As an example of the use of the algorithm consider finding the minimum of the function given by

$$y = x^2 - 1.1x + 2.$$

It is a trivial exercise to solve this using differentiation to find the minimum is at x = 0.55 with y = 1.6975. Figure 3-2 shows a graph of the curve for this function on the interval from zero to one. Suppose the user selected 0.9 as the staring point with a pulse size of 0.1. The resulting y at this point is 1.82. The next step is to subtract one pulse from the x value yielding 0.8 as the new value. The resulting y at this point is 1.76 which is an improvement from the previous point. The algorithm would now keep subtracting pulses and testing the results until it reaches x = 0.4. At this point the previous result (i.e., at x = 0.5) is 1.7 and the new result is 1.72. We now know that the minimum is between the points x = 0.4 and x = 0.6. Now the algorithm

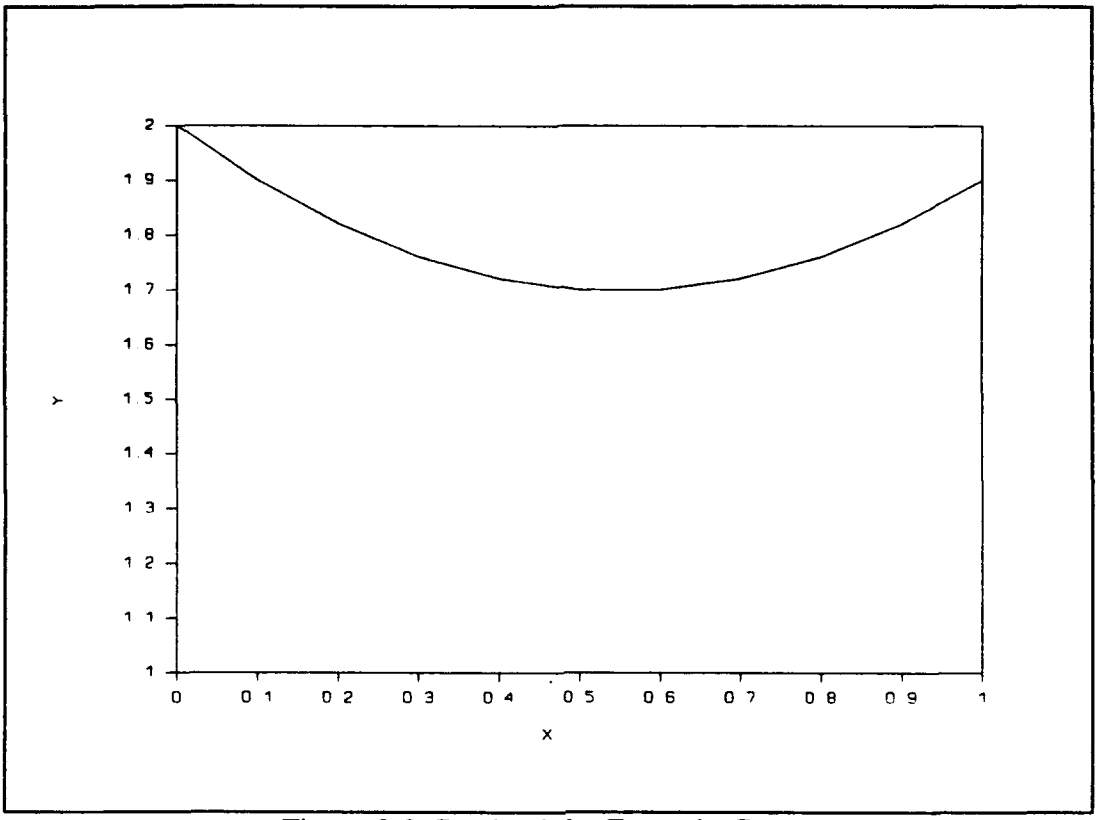

Figure 3-2 Graph of the Example Curve.

restarts at x = 0.5 (since it yielded the lowest value for y of the points tested so far) with the pulse size reduced to 0.05. The first step tries the point x = 0.45 which gives y = 1.7075. Since this is not an improvement over x = 0.5 an additive pulse is tried (i.e. x = 0.55). This is of course the minimum value for the function, however, the algorithm will continue to make a few more excursions around this point before the pulse size is reduced below the minimum and the algorithm terminates.

3. Description of Major Rules

This section describes the important rules that are used in the program single.ari to accomplish the algorithm given in the previous section. For a complete listing of the source code for this program see Appendix A.

a. Rule- optimize_single

This rule is the main module for the program single.ari. It first calls the rule init_vars to get the initial values for the variable name, the pulse size, the minimum pulse size, and the two starting values from the user. The rule minimize is then called with the first value to find the variable value with the minimum cost. Once minimize is complete, the results from the first value are printed and then destroyed. Minimize is again called, but this time with the second value as the starting point. In order to maximize the chances of finding two distinct minimums, if they exist, the two points should be at opposite ends of the range of reasonable values for the given variable.

```
/************************ Rule optimize_single ************************/

optimize_single :-
    init_vars(Pulse,Value1,Value2),

/* Test at the initial point       */

    get_cummd(Value1,0,Cummd),      /* Find CUMMD at first point */
    minimize(Value1,Pulse,0),
    output_message('The first minimum is at '),

/* Test at the second point  */

    abolish(result/3),              /* Destroy results from first point */
    abolish(size/1),

    asserta(size(Pulse)),           /* Return to the initial pulse size */
    get_cummd(Value2,0,NewCummd), /* Find CUMMD at second point */

    minimize(Value2,Pulse,0),
    output_message('The second minimum is at ').
```

b. Rule- minimize

This rule is used to find the value of the test variable that minimizes project cost using the algorithm given in the previous section.

```
/************************ Rule minimize ******************************/

/* If the exit condition is met end the rule  */
 minimize(Value,Pulse,I) :-
    exit_condition, !.
```

```
/*  Otherwise continue to execute the search algorithm  */

minimize(Value,Pulse,I) :-
     minval(Minval),
     maxval(Maxval),
     NewI is I + 1,

     /* Pulse in the negative direction  */
     NewValue is Value - Pulse,

     /* Ensure that the new value is not less then the minimum */
     range_check(NewValue,Maxval,Minval,ScaledVal),

     /* Retrieve old result and run the model to find new result */
     result(I,Value,OldC),
     get_cummd(ScaledVal,NewI,NewC),

     /* If the negative pulse was an improvement then set value of Pulse to
      * Pulse * (-1), otherwise pulse in the positive direction
      * NOTE-- the ifthenelse predicate is a built-in predicate in Arity prolog and is not
      * a standard prolog predicate.  Its use is if the condition is true then the first
      * predicate is executed, otherwise the second predicate is executed.
      */
     ifthenelse(NewC =< OldC,pulse_neg(Pulse),
                             pulse_pos(Value,Pulse,NewI)),

     /* Continue in the direction of improvement starting at the last
      * point tested by calling the rule find_min
      */
     size(NewPulse),
     result(NewI1,NewValue1,NewC1),
     find_min(NewValue1,NewPulse,NewI1,NewC1,OldC),

     /* Retract the result from the point where improvement was no          * longer
      * noted. */
     retract(result(A,B,C)),

     /* Retrieve the result from the last point of improvement and          * assert it
      * as the new minimum.
      */
     result(MinI,MinValue,MinCummd),
     asserta(min(MinValue,MinCummd)),

     /* Halve the Pulse size and return to the beginning of the rule */
     SmallerPulse is Pulse * 0.5,
     abolish(size/1),
     asserta(size(SmallerPulse)),
     minimize(MinValue,SmallerPulse,MinI).
```

c. *Rule- pulse_neg*

This rule is called whenever an initial negative pulse results in improvement over the starting value. Since the value of the variable Pulse is always added to the current value to calculate the new value, the sign of Pulse is changed to negative to give the effect of subtraction.

```
/************************ Rule pulse_neg ******************************/

pulse_neg(Pulse) :-
    NewPulse is Pulse * -1,
    abolish(size/1),
    asserta(size(NewPulse)).
```

d. *Rule- pulse_pos*

This rule is called when the initial negative pulse does not result in improvement. First the result from the negative pulse is retracted from the database, then a positive pulse is added to the previous value of the variable. Rule get_cummd is then called to find the value of CUMMD at the new point.

```
/************************ Rule pulse_pos ******************************/

pulse_pos(Value,Pulse,I) :-
    minval(Minval),
    maxval(Maxval),
    retract(result(I,V,X)),
    NewValue is Value + Pulse,
    range_check(NewValue,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,I,Cummd).
```

e. *Rule- find_min*

This rule is used to continue to pulse in the direction given by the sign of Pulse until improvement in the value of CUMMD is no longer noted. The rule will also terminate if it attempts a positive pulse at the upper limit or a negative pulse at the lower limit since values past these points are not valid.

```
/************************ Rule find_min ******************************/

/* If the new CUMMD is more than the old CUMMD then exit the rule */
find_min(Value,Pulse,I,NewC,OldC) :-
    NewC > OldC,!.
```

```
/* If the current pulse is positive and the variable is at the maximum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    maxval(Maxval),
    Pulse > 0,
    Value == Maxval,!.

/* If the current pulse is negative and the variable is at the minimum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    minval(Minval),
    Pulse < 0,
    Value == Minval,!.

/* Otherwise pulse in the current direction of the variable Pulse and
 * then run the model to get the new value of CUMMD.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    minval(Minval),
    maxval(Maxval),

    /* Pulse in the previously set direction of improvement */
    NewVal is Value + Pulse,

    /* Ensure that the new value is between the user supplied minimum
     * and maximum value
     */
    range_check(NewVal,Maxval,Minval,ScaledVal),

    /* Get the new value of CUMMD and iterate the rule through the
     * use of tail recursion.
     */
    NewI is I + 1,
    get_cummd(ScaledVal,NewI,Cummd),
    find_min(ScaledVal,Pulse,NewI,Cummd,NewC).
```

f. Rule- exit_condition

This rule is used to determine if the algorithm should terminate. It succeeds if the pulse size is reduced below the user set minimum pulse size.

```
/********************** Rule exit_condition **********************/

exit_condition :-
    size(Pulse),
    minsize(MinPulse),!,
    Pulse < MinPulse.
```

4. Program Operation

To initiate the program be sure you are in the directory containing the prolog interpreter, the files listed in Section B, and the source file for single.ari. Start the prolog interpreter by typing:

C> **API** (all user required inputs appear in boldface type)

Once you get the Arity Prolog prompt (?-) type:

?- **[single].**

The interpreter should respond with a "yes" and return to the prompt (if it responds with "no" then exit prolog by typing **halt.** and make sure the file single.ari exists in the current directory then restart the procedure from the beginning). Type:

?- **optimize_single.**

The program will now ask a series of seven questions (shown below with sample responses). Enter the desired response and then press enter. The first question will be:

What variable do you want to optimize? **DEVPRT**

The variable name must be typed in caps and be spelled exactly like in the model (if a mistake is noticed after pressing enter press **control-C**, then **halt**, and then restart from beginning).

The next question will be:

What is the maximum value for the variable? **0.99**

This is the upper limit on the values to be tested by the program.

The third question asked will be:

What is the minimum value for the variable? **0.3**

This is the lower limit on the values to be tested by the program.

The fourth question will be:

What is your desired pulse size? **0.05**

This is the size of the pulse that will be initially added to and subtracted from the current variable value to get the new value for testing. This number should be large enough to a see noticeable change in the value for CUMMD at the new points. As the minimum is approached the size is reduced by the program to refine the solution.

The fifth question asked is:

What is your minimum pulse size? **0.005**

This the minimum pulse size used determine when to end the program. When the pulse size is reduced below this value the program is terminated.

The sixth question asked is:

What is the value for the first test point? **0.9**

This must be a number between the maximum and the minimum values input earlier by the user (e.g., 0.99 and 0.3 for this example) and should be near the maximum point to ensure the entire range is searched.

The final question is:

What is the value for the second test point? **0.5**

The same restrictions apply as above and this value should be near the minimum for the same reasons.

After these questions have been answered the program will begin the optimization. The time to complete the process depends on the variable being tested and can be quite long. When the program is complete the results can be found in the file summary.dat. An example of the listings found is this file is:

```
DEVPRT= 0.9
1.  CUMMD= 1656.71
DEVPRT= 0.85
2.  CUMMD= 1597.04
```

The minimum point is identified in the file by a header being printed followed by the variable value and CUMMD for the minimum point.

D. EXPERT MODULE TABLE.ARI

As stated in Chapter II, an expert system module was designed and implemented that would allow for the optimization of the variable "Planned Fraction of Manpower for QA" (TPFMQA). The purpose of the module TABLE.ARI is to improve on the previous design in two ways. The first is to make the routine generic so it can be used to optimize any "table function" with respect to total project cost. The second purpose is to increase the efficiency and performance of the optimization by using a more sophisticated algorithm. The "pattern search" technique, developed by Hooke and Jeeves [Ref. 6], appeared to be well suited to this application and was chosen for implementation.

1. Algorithm Description

The pattern search technique is based on the assumption that any adjustments to a set of independent variables which have been successful during early experiments will be worth trying again [Ref. 6:p. 145].

> Although the method starts cautiously with short excursions from the starting point, the steps grow with repeated success. Subsequent failure indicates the shorter steps are in order, and if a change in direction is required the technique will start over again with a new pattern. In the vicinity of the optimum point the steps become very small in order to avoid overlooking any promising direction [Ref. 6:p. 145].

a. Establishing a Pattern

The pattern begins at the base point $\mathbf{b}_1$, which is the initial set of values entered by the user. The user chooses a step size s_i for each independent variable. For this program the same step size is used for each independent variable[4]. First the

---

[4]For this application the independent variables are the values in the table function being tested.

simulation is run at the base point $\mathbf{b}_1$ and the value for CUMMD is read. Next the point $\mathbf{b}_1 + \mathbf{s}_1$ (the step size added to the first value in the table function) is used in the simulation. If the value for CUMMD is better at this point then at $\mathbf{b}_1$ we call $\mathbf{b}_1 + \mathbf{s}_1$ the temporary head $\mathbf{t}_{11}$. The double subscript shows that the first pattern is being developed and that the first variable has been perturbed. If $\mathbf{b}_1 + \mathbf{s}_1$ is not better than $\mathbf{b}_1$ it is forgotten and the point $\mathbf{b}_1 - \mathbf{s}_1$ is tried. If this new point is better than $\mathbf{b}_1$ it becomes the temporary head; otherwise $\mathbf{b}_1$ is designated the temporary head. To summarize $\mathbf{t}_{11}$ is the best (lowest value for CUMMD) of (1) $\mathbf{b}_1 + \mathbf{s}_1$; (2) $\mathbf{b}_1 - \mathbf{s}_1$; or (3) $\mathbf{b}_1$ [Ref. 6:p 146].

Perturbation of each independent variable is carried out in the same manner, each time about the previous temporary head. For example the second independent variable is perturbed about $\mathbf{t}_{11}$, the third about $\mathbf{t}_{12}$, etc. When all of the variables have been perturbed the last temporary head $\mathbf{t}_{1k}$ is designated the second base point $\mathbf{b}_2$ [Ref. 6:p. 147].

b. Pattern Moves

The original base point $\mathbf{b}_1$ and the newly determined base point $\mathbf{b}_2$ together form the first pattern. Reasoning that if a similar exploration were conducted from $\mathbf{b}_2$ the results will likely be the same the pattern is jumped to the new temporary head $\mathbf{t}_{20}$ as follows:

$$\mathbf{t}_{20} = \mathbf{b}_2 + (\mathbf{b}_2 - \mathbf{b}_1) = 2\mathbf{b}_2 - \mathbf{b}_1$$

The subscript 20 indicates that a second pattern is being built, but the variables have not yet been perturbed. Local exploration is conducted around $\mathbf{t}_{20}$ in order to determine the next direction for the pattern. The pattern continues to grow with each success in a given direction moving the pattern more rapidly toward the optimum point [Ref. 6:pp. 147-148].

At some point in the search none of the perturbations around an initial temporary head $\mathbf{t}_{j0}$ will improve the result, however, if the result at $\mathbf{t}_{j0}$ is less than at the previous base $\mathbf{b}_{j-1}$ then the pattern will maintain its direction, but without any growth. If the result at $\mathbf{t}_{j0}$ is not an improvement over $\mathbf{b}_{j-1}$ then the pattern is destroyed. This could mean that the optimum point has been found, or the pattern has jumped past it. In either case new maneuvers are required [Ref. 6:p. 148].

c. Pattern Reestablishment

Since the pattern cannot be continued from temporary head $\mathbf{t}_{j0}$ it must be abandoned and rebuilt starting from base $\mathbf{b}_{j-1}$. Since a new pattern is being built, base $\mathbf{b}_{j-1}$ is renamed $\mathbf{b}_j$ and it becomes the new temporary head $\mathbf{t}_{j0}$ for local exploration. If these perturbations result in a better value for CUMMD than found at $\mathbf{t}_{j0}$ the final temporary head is designated $\mathbf{b}_{j+1}$ and a new pattern is formed. If however, improvement is not found then the step size is halved and the perturbations are repeated in an attempt to locate a new pattern [Ref. 6:p. 149].

d. Ending the Search

If after the reduction of the pulse size an improved solution is found then the pattern is reestablished and moves off in the new direction. However, if reducing the step size does not find a better result then the step size is halved once again. The reductions continue until the step size falls below the user set minimum size. At this point the current base is considered to be the optimum point and the search is ended [Ref. 6:p. 150].

2. Example Using the Algorithm

As an example consider using the pattern search algorithm to find the minimum project cost to be obtained from adjusting the variable TPFMQA. Starting with a uniform QA distribution of 15% designated b_1 the initial value for CUMMD is 1656.71 man-days. The next step is to perturb each of the ten values in the table. If a 0.05

step size is used the first value becomes 0.2 while the rest remain at 0.15. This distribution results in a value of 1597.04 man-days for CUMMD, so 0.2 is retained and the next value is perturbed. In this case neither raising or lowering the table value improves CUMMD so the 0.15 is retained. The perturbations continue for the eight remaining values and at the end of one cycle the values for TPFMQA are:

0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15

and CUMMD has been reduced to 1551.01 man-days. This point is the final temporary head t_{110} and since it results in an improved value for CUMMD it becomes the second base b_2. The pattern is now jumped to t_{20} of:

0.25/0.15/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15

in accordance with the algorithm. Figure 3-3 shows the TPFMQA distributions for b_1 and b_2 and the resulting distribution for t_{20} after the pattern jump.

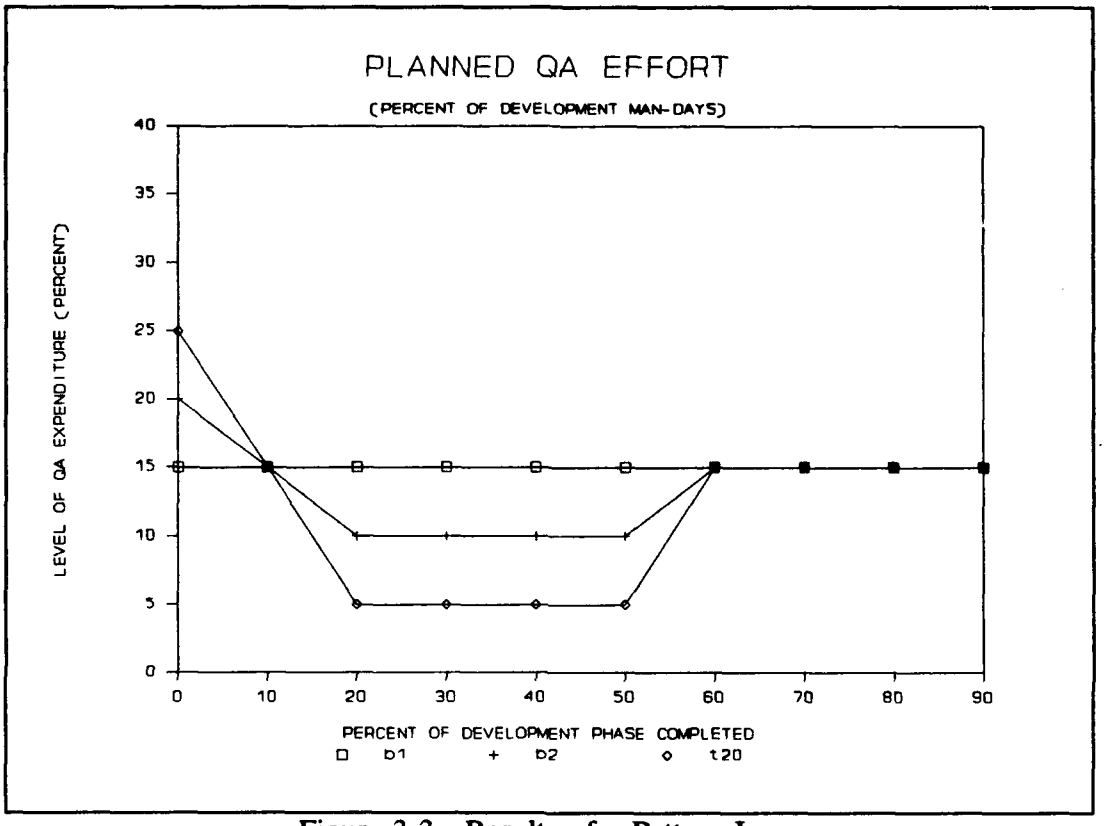

Figure 3-3. Results of a Pattern Jump

This pattern of perturbations followed by pattern jumps continues for three cycles until the temporary head t_{610} fails to show improvement over base b_6. At this point b_6 is designated t_{70} and local explorations are conducted around it. These also fail to show any improvement so the pulse size is reduced to 0.025 and the explorations are again conducted around t_{70} without improvement in CUMMD. The pulse size is reduced further to 0.0125 and the local explorations are repeated. This also fails to produce an improvement and another reduction of the pulse size decreases it below the minimum value of 0.01 and the search is terminated. The value for TPFMQA with the minimum CUMMD is:

0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02.

Figure 3-4 shows the resulting movment of the pattern from the start at b_1 to the minimum at b_6.

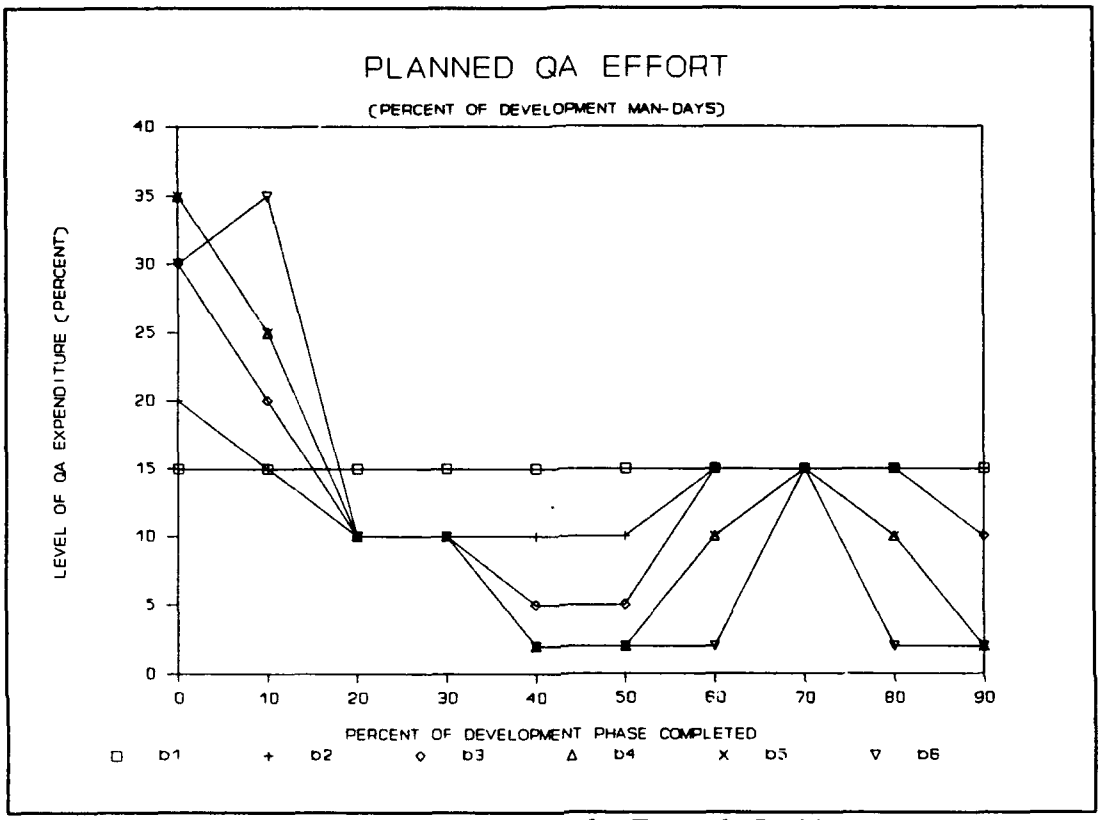

Figure 3-4. Pattern Moves for Example Problem

3. Description of Major Rules

This section describes the important rules that are used in the program table.ari to accomplish the pattern search algorithm. For a complete listing of the source code for this program see Appendix B.

a. Rule - pattern_search

This rule first checks to see if the exit condition has been met and if so the rule is terminated. The remainder of the rule is separated into three pieces to handle the

three separate conditions delineated in the algorithm. The first section handles the case where perturbations about the temporary head lead to an improvement over the current base so the pattern is jumped to a new temporary head. The second section handles the situation where the above case fails to show improvement so the previous base becomes the new temporary head for local explorations. The third section is executed if both of the above cases fail and the pulse size must be reduced before another set of perturbations is performed.

```
/************************ Rule pattern_search ************************/

/* Test if the exit condition is true  */
pattern_search :-
    exit_condition.

/* Case where final temporary head is better than previous base so temporary head
 * becomes the new base2 and the pattern is jumped.
 */

pattern_search :-
    /* NOTE-- the snip operators ([!,!]) are not standard prolog.  Their purpose is
     * similar to the cut operator (!) except that they only prevent backtracking into the
     * statements inside the snips and they do not have the side effect of preventing
     * execution of other predicates of the same name as the one where they were
     * executed.
     */

    /* Retrieve the value for CUMMD at the current base and the size of the table from
     * the database.
     */
    [!basecummd(OldCummd),
    tabsize(TableSize),

    /* Perturb each of the variables in the table  */
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store values for each variable to the list temp */
    result(NewCummd)!], /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd),

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only execute if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve current values for the temp head */
```

```
    /* Set base2 equal to the temporary head and set the value basecummd to
     * NewCummd.
     */
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
    asserta(basecummd(NewCummd)),

    /* Jump the pattern to new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.


/* Case where temp head is not better than base so try local exploration about base.
 */
pattern_search :-

    /* First test to see if already at base1.  If not retreat to base1 and perform local
     * explorations.
     */
    [!not(already_at_base1),
    output_message('Retreat to Base for Local Exploration'),
    retrieve_base1_values,

    /* Retrieve the table size and value for CUMMD at the base from the database.
     */
    tabsize(TableSize),
    basecummd(OldCummd),

    /* Perturb each of the variables in the table  */
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store values for each variable to the list temp */
    result(NewCummd)!], /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd),

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only execute if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve current values for the temp head */

    /* Set base2 equal to the temporary head and set the value basecummd to
     * NewCummd.
     */
```

```
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
    asserta(basecummd(NewCummd)),

    /* Jump pattern to the new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.

/* Case where local explorations about base1 do not result in improvement so reduce
 * pulse size and start pattern over from base1.
 */

pattern_search :-
    output_message('Reduce Pulse Size'), /* Annotate summary.dat  */
    retrieve_base1_values,

    /* Reduce pulse size and start pattern search over  */
    size(Pulse),
    NewPulse is Pulse * 0.5,
    retract(size(Pulse)),
    asserta(size(NewPulse)),
    pattern_search.
```

b. Rule - perturb_variables

This rule is used to retrieve the values needed for the perturbations from the database and then call do_perturbation to perform the actual variable perturbation. The rule is called iteratively by the rule foriterate which is used to create a for loop structure similar to C or Pascal.

```
/********************** Rule perturb_variables *********************/

perturb_variables :-
    /* Retrieve the current value for the iteration counter, the value
     * of the table variable at the counter position, and the pulse size from database.
     */
    counter(Iter),
    variable(Iter,Val),
    size(Pulse),!,

    do_perturbation(Iter,Val,Pulse).
```

c. *Rule - do_perturbations*

This rule is called by perturb_variables and is used to perform the perturbations of the table function values as detailed in the algorithm description. Three cases are handled: (1) a positive pulse is added to the current value; (2) a negative pulse is subtracted from the current value; or (3) the value is returned to its original value if the previous two cases fail.

```
/********************** Rule do_perturbations **********************/

/* Add a positive pulse to the variable value and then run the model to test for
 * improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!min(Minval),
    max(Maxval),

    /* Apply the positive pulse and run the model with the new values */
    NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
    asserta(result(NewC)),!.

/* If the above rule fails then apply a negative pulse and then run the model to test for
 * improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!min(Minval),
    max(Maxval),

     /* Apply a negative pulse and run the model with the new values */
    NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
```

```
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
    asserta(result(NewC)),!.

/* If both of the above rules fail then return the table variable to its original value.
 */
do_perturbation(Iter,Val,Pulse) :-
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)).
```

d. Rule - jump_pattern

This rule accomplishes the pattern jump as listed in the algorithm description.

```
/********************** Rule jump_pattern **************************/

jump_pattern :-
    /* Retrieve the current base1 and base2 value lists */
    base1(Base1Vallist),
    base2(Base2Vallist),

    min(Minval),
    max(Maxval),
    abolish(variable/2),
    asserta(count(0)),
    /* Calculate the new values for the temporary head */
    calc_new_vals(Base1Vallist,Base2Vallist,Minval,Maxval),
    abolish(count/1),

    /* Run the model at the temporary head position.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    abolish(result/1),
    asserta(result(Cummd)),
    output_cummd(Cummd,0).
```

4. Program Operation

First at the DOS prompt start the prolog interpreter by typing **API** as for the previous program. When the interpreter returns its prompt type:

?- **[table].**

Once the interpreter responds with "yes" and returns the prompt acknowledging that the file has been loaded type:

?- **optimize_table.**

The program will now ask a series of questions to get the intial input from the user. The first question is:

What table variable do you want to optimize? **TPFMQA**

The name entered is the name of the table function variable as it appears in the systems dynamics model in all capital letters.

The next question asked is:

How many values are in your table? **10**

This is the number of values in the table function being tested.

The system will now ask the user for the maximum and minimum values for the variable and the pulse size and the minimum pulse size just as for the previous program. Once these questions are answered the user is asked to enter the starting values for the table function one value at a time. An example of one such input is:

Enter Table Value 1. **0.325**

The other nine values are now entered in the same manner. After the last value is entered (value 10 in this case), the program begins by executing the model with the initial values and then begins the pattern search algorithm. Several simulations are required to find the optimal distribution for any table function so execution of the program may take as long as a few hours. Again, an annotated listing of the results will be stored in the file summary.dat for post-run analysis.

E. EXPERT MODULE DEPTH.ARI

This program is the first of two that are intended to optimize two variables in the systems dynamics model simultaneously. The program depth.ari uses the depth-first search scheme to accomplish the optimization.

1. Algorithm Description

To accomplish the depth-first search the operators chosen were (1) to call the rule optimize_table to optimize the table function values; and (2) to call the rule optimize_single to optimize the single valued variable. The start state is defined as the initial values for both variables as entered by the user and the resulting value for CUMMD. The search begins by applying the operator optimize_table which will continue until the optimal distribution for the table function is found. Next the operator optimize_single is applied to find the optimal value of the single valued variable.

2. Example Using the Algorithm

Appendix E shows the result of executing a depth-first search on the two variables TPFMQA and DEVPRT, the variables for allocating effort to QA and testing in the model. The starting values are:

TPFMQA = 0.325/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4

DEVPRT = 0.85.

The initial value for CUMMD at this point is 2093.08 man-days. The rule optimize_table is now called to execute the pattern search algorithm on TPFMQA with DEVPRT held constant at 0.85. The optimal values for TPFMQA are found to be:

0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01.

The resulting value for CUMMD is 1469.79 man-days. The rule optimize_single is now called to optimize DEVPRT with the values for TPFMQA held constant at the optimal point. The optimal value for DEVPRT is found to be:

DEVPRT = 0.85

with the resulting value for CUMMD unchanged from the previous result.

3. Description of Major Rules

Since this program uses the rules optimize_table and optimize_single discussed in the previous sections, the only rule that needs to be discussed here is the one that executes the depth-first search.

a. Rule - depth_search

This rule is used to implement the depth-first search by first calling the rule optimize_table and then calling the rule optimize_single. The order for applying the operators was chosen to best suit the variables TPFMQA and DEVPRT[5], however, the order may be reversed for a different set of variables by simply switching the order of the calls to the rules optimize_table and optimize_single.

```
/********************** Rule depth_search ***************************/

depth_search :-
    init_vars,
    optimize_table,
    optimize_single,
    halt.
```

4. Program Operation

First start the prolog interpreter from the DOS prompt by typing **API**. At the prolog prompt type:

?- **[depth].**

When the prolog prompt is returned type:

?- **depth_search.**

The user will now be asked a series of questions that are a combination of the questions from the two previous programs. Execution of the depth-first search algorithm begins after the last question is answered. The time requirements are similar to table.ari and the file summary.dat will again hold an annotated listing of the results.

---

[5]An experiment was run with the order reversed and the final value for CUMMD was 50 man-days higher than the value found using this order.

F. EXPERT MODULE BREADTH.ARI

The purpose of this program is accomplish two-variable optimization using the breadth-first search algorithm.

1. Algorithm Description

The breadth-first search uses the same state description as for depth-first search, but the operators are slightly different. In this case the two operators are to call the rule perturb_table_variable to do one cycle of pulse adjustments on the table function values, and to call perturb_single_variable to apply adjustments to the single valued variable. The search begins by applying the operator perturb_table_variable to find the best value for CUMMD after one cycle of adjustments. Next the operator perturb_single_variable is applied to adjust the single valued variable holding the table function values constant. Now the operator perturb_table_variable is again applied using the values previously derived as a starting point. This is followed by another call to perturb_single_variable and this pattern continues until the application of both operators in a cycle fails to yield any improvement in the value for CUMMD. At this point the pulse size for both variables is halved and the algorithm starts over from the beginning.

2. Example Using the Algorithm

Appendix F shows the result of executing a breadth-first search on the two variables TPFMQA and DEVPRT. The starting values are:

TPFMQA = 0.325/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4

DEVPRT = 0.85

The value for CUMMD at this point is 2093.08 man-days. First the rule perturb_table_variable is called to perform one cycle of perturbations on the table

variable values with the values for DEVPRT held constant. The resulting values are (initial pulse size is 0.05):

TPFMQA = 0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35.

The value for CUMMD at this point is 1813.24 man-days. Next the rule perturb_single_variable is called to find the optimal value for DEVPRT given the above value for TPFMQA. After adjustments the new value is (initial pulse size also 0.05):

DEVPRT = 0.9.

This results in a value for CUMMD of 1801.32 man-days. This cycle is repeated for several iterations until the point where no better value for CUMMD is found after adjusting both variables. The pulse size for both variables is now reduced to 0.025 and the algorithm repeats. Once the pulse size is reduced below the minimum set by the user the algorithm terminates.

3. Description of Major Rules

This program uses many of the rules already described in previous sections so only those rules peculiar to this application will be discussed.

a. Rule- breadth-search

This rule first queries the user for the parameters and starting values for the variables and then executes the model to find the inial value for CUMMD. Finally the rule do_breadth_search is called to execute the breadth-first search.

```
/********************** Rule breadth_search ************************/

breadth_search :-
    init_vars,
    initial_run,
    do_breadth_search.
```

b. Rule- do_breadth_search

The purpose of this rule is to execute the breadth-first search for the two variables by first calling the rule perturb_table_variable and then calling the rule

perturb_single_variable. At the completion of these two rules the value of CUMMD is checked against the result from the previous cycle. If the adjustments resulted in improvements then that section of the rule succeeds and the rule calls itself again. If improvement was not found then the pulse size for both variables is halved and the rule calls itself again. The rule terminates when the exit condition of the pulse sizes being less than the minimum is true.

```
/********************** Rule do_breadth_search **********************/

/* If the exit condition has been met then exit the rule */
do_breadth_search :-
    exit_condition.

/* Otherwise carry out the breadth-first search by first applying the
 * operator perturb_table_variable to run one cycle of adjustments on the table variable
 * and then applying the rule perturb_single_variable to test the constant. If a better
 * value for CUMMD is found then the rule calls itself to continue the algorithm.
 */
do_breadth_search :-
    /* retrieve the table size and previous value for CUMMD from the
     * database.
     */
    [!tabsize(TableSize),
    result(OldCummd),

    /* Perturb each of the values in the table function */
    foriterate(perturb_table_variables,TableSize),

    /* Perturb the value of the single valued constant */
    test(Val),
    perturb_single_variable(Val),

    /* Retrieve the new value for CUMMD from the database and compare
     * it with the previous value.
     */
    result(NewCummd)!],
    NewCummd < OldCummd,

    /* If the above test succeeds then store the current values in the
     * database as the new minimum point and start another cycle.
     */
    store_min_values,
    do_breadth_search.

/* If adjustments to both variables fail to find a better value for CUMMD then reduce
 * both pulse sizes and restart the search.
 */
```

```
do_breadth_search :-
    /* Retrieve the current pulse sizes from the database */
    t_size(TPulse),
    s_size(SPulse),

    /* Halve both pulse sizes and store new values in the database */
    NewTPulse is TPulse * 0.5,
    NewSPulse is SPulse * 0.5,
    abolish(t_size/1),
    abolish(s_size/1),
    asserta(t_size(NewTPulse)),
    asserta(s_size(NewSPulse)),

    /* Restart the search with the smaller pulse sizes */
    do_breadth_search.
```

c. Rule- perturb_table_variable

This rule is the same as the rule perturb_variables described in Section C.

d. Rule- perturb_single_variable

This rule is similar in function to the rules to perturb the table variable values in that it has three parts. First a positive pulse is added to the current value and the model is run to see if the resulting value for CUMMD is less than the previous value. If not a negative pulse is applied and the model is executed again to test for improvement in CUMMD. If neither of these steps succeeds than the variable is returned to its original value.

```
/****************** Rule perturb_single_variable ********************/

/* Add a positive pulse to the variable and then run the model to
 * test for improvement. If the value is already at the maximum
 * before the positive pulse then the rule fails and the next case
 * is executed.
 */
perturb_single_variable(Val) :-
    [!s_min(Minval),
    s_max(Maxval),
    s_size(Pulse)!],

    /* If the variable is already at the maximum value then fail */
    Val \== Maxval,

    /* Add a positive pulse and run the model */
    [!NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
```

```
    s_get_cummd(ScaledVal,1,NewC),
    result(OldC)!],

    /* If the new CUMMD is less than the old then store the result
     * in the database.
     */
    NewC < OldC,
    abolish(result/1),
    asserta(result(NewC)).
/* If the above rule fails then apply a negative pulse and run
 * the model again.  If the variable is already at the minimum value
 * before the negative pulse then the rule fails and the next case is
 * executed.
 */
perturb_single_variable(Val) :-
    [!s_min(Minval),
    s_max(Maxval),
    s_size(Pulse)!],

    /* If the variable is already at the minimum value then fail */
    Val \== Minval,

    /* Apply the negative pulse and run the model. */
    [!NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    s_get_cummd(ScaledVal,2,NewC),
    result(OldC)!],

    /* If the new CUMMD is better than the old CUMMD then store the
     * result in the database.
     */
    NewC < OldC,
    abolish(result/1),
    asserta(result(NewC)).
/* If both the above cases fail then return the variable to its
 * original value.
 */
perturb_single_variable(Val) :-
    abolish(test/1),
    asserta(test(Val)).
```

4. Program Operation

First start the prolog interpreter from the DOS prompt by typing **API**. At the prolog prompt type:

?- **[breadth].**

When the prolog prompt is returned type:

?- **breadth_search.**

This same questions will be asked as were for depth.ari and then the program will begin execution of the breadth-first algorithm. Upon completion of the run the results will be in the file summary.dat.

IV. EXPERIMENTAL RESULTS

A. THE NASA DE-A SOFTWARE PROJECT

To test the performance of the expert system simulation model its results were compared to the results of an actual software project. The project involved development of the software for the DE-A satellite designed and implemented by the Systems Developments Section of NASA's Goddard Space Flight Center (GSFC) at Greenbelt, Maryland. The software system was designed to processes telemetry data and provide altitude determination and control for the DE-A satellite.

The initial estimates for the project were as follows:

project size	16,000 delivered source instructions (DSI)
development cost	1,100 man-days
completion time	320 days

Quality assurance allocation was set to 30% of the total development effort and 15% of the total project effort was allocated to testing. The final project statistics were:

project size	24,000 DSI
development cost	2,200 man-days
completion time	380 days

Figure 4-1 shows the actual distribution for quality assurance (as a percentage of total development effort) used in the DE-A project.

The NASA DE-A project simulation was used as a basis for the experiments conducted as part of this thesis. The first set of experiments involves using the depth-first and breadth-first modules to find the optimal allocation for TPFMQA and DEVPRT in the DE-A project. The second set of experiments is a series of tests designed to test the sensitivity of the results from the first experiments to changes in certain project parameters.

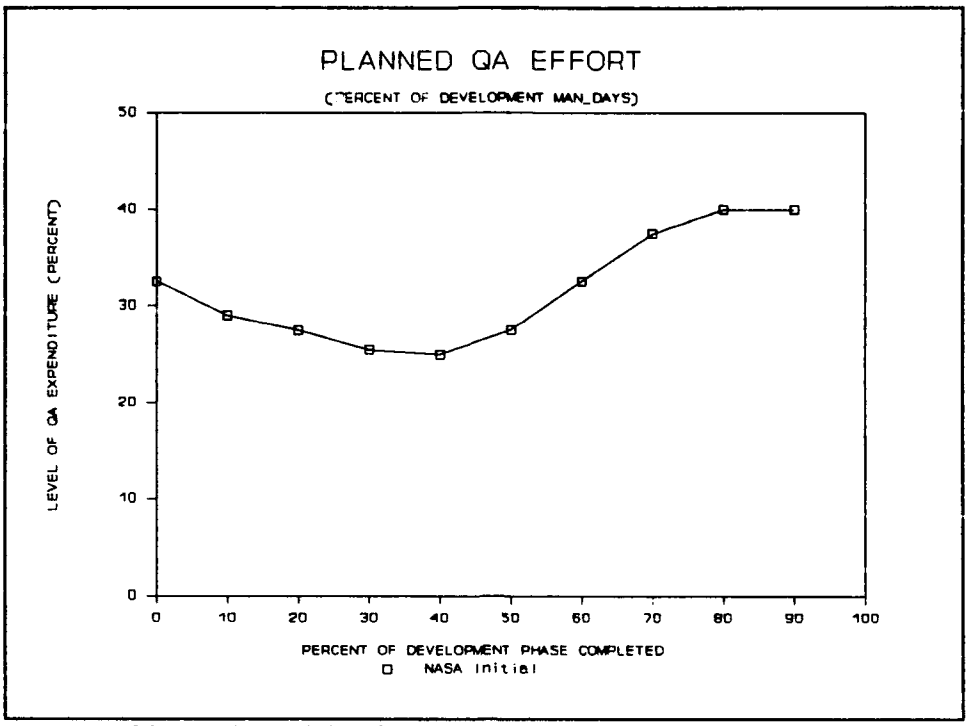

Figure 4-1. QA Distribution for NASA DE-A Project

DEVPRT = 0.85

NASA DE-A INITIAL DATA

Results:

Completion time	387	Days
Total Man-Days	2093.08	Man-Days
Total Dev't MD	1822.75	Man-Days
Design & Code	1298.50	Man-Days
QA MD	524.25	Man-Days
Total Testing MD	270.33	Man-Days

B. RESULTS OF DEPTH-FIRST SEARCH

The program depth.ari was used to find the optimal allocation for quality assurance and testing in the DE-A project. The actual values used by NASA namely:

TPFMQA = 0.325/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4

DEVPRT = 0.85

were used as the starting point for the search. The starting pulse size was set to 0.05 for both variables with a minimum pulse size of 0.005. Appendix E contains a complete listing of the results of this simulation. Figure 4-2 shows the resulting optimal distribution for TPFMQA with the figures that follow showing the breakdown of total effort for each of the lifecycle phases. The figure shows that quality assurance efforts are most beneficial in the early stages of the development phase. The reason for this is that errors made during the design phase will multiply to become errors in the code, documentation, etc if they are not detected and corrected as early as possible. The results also show allocation of effort for quality assurance during coding is not as cost effective as deferring the correction of the error until the testing phase. Using the optimal QA distribution the total effort was reduced by 623 man-days and the completion time reduced by 54 days. The optimal value for DEVPRT did not change from the actual value used by NASA.

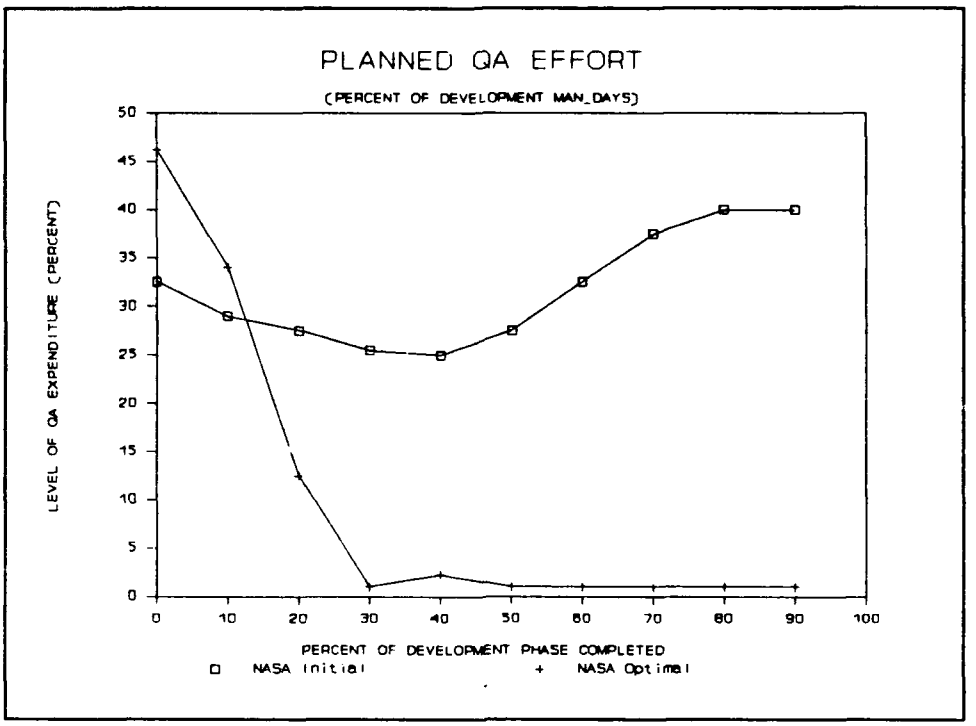

Figure 4-2. QA Distribution After Depth-First Search

RESULTS OF DEPTH-FIRST SEARCH

Results:

Completion time	333	Days
Total Man-Days	1469.79	Man-Days
Total Dev't MD	1269.28	Man-Days
Design & Code	1083.72	Man-Days
QA MD	185.55	Man-Days
Total Testing MD	200.52	Man-Days

	NASA Initial	NASA DEPTH-FIRST Optimal
DEVPRT	0.85	0.85
CUMMD	2093.08	1469.79

C. RESULTS OF BREADTH-FIRST SEARCH

The program breadth.ari was also run on the DE-A project to use a breadth-first search technique to find the optimal levels for TPFMQA and DEVPRT. Appendix F contains a complete listing of the results of this run. Figure 4-3 shows the optimal distribution for TPFMQA arrived at using this search technique. The data that follows the figure gives a breakdown of the total development effort into the major lifecycle phases. This technique did find values for TPFMQA and DEVPRT that reduced the project cost and decreased the completion time, however, the improvement was not as substantial as was found using depth-first search. Additionally, it can be seen by comparing the results in Appendix F with those in Appendix E that the depth first search was much more efficient in arriving at the minimum value. For these reasons the program depth.ari was chosen for use in the experiments that follow.

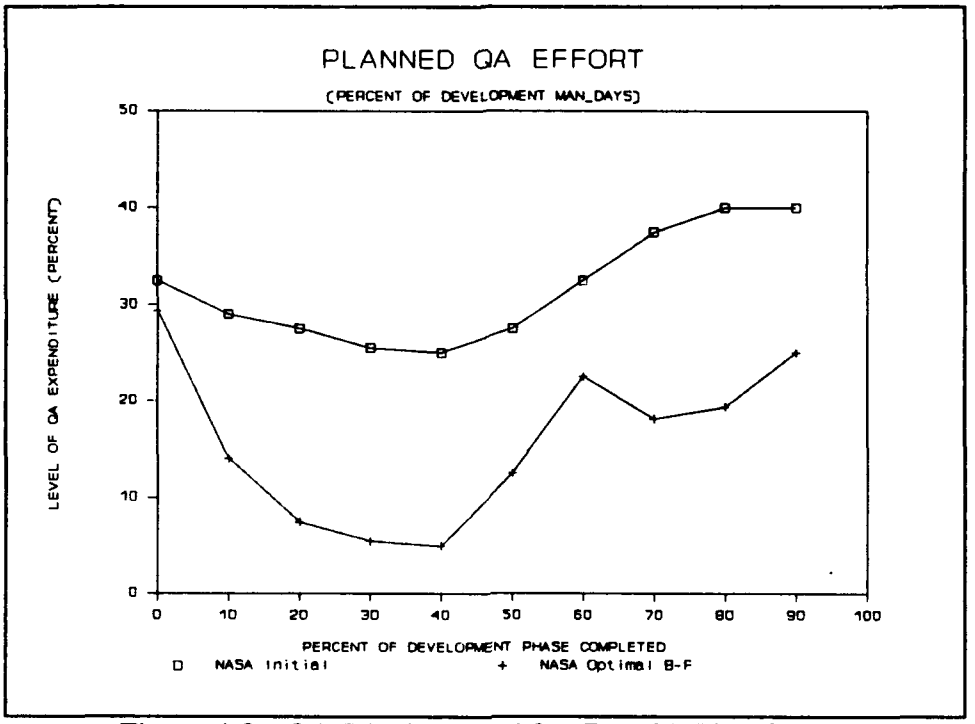

Figure 4-3. QA Distribution After Breadth-First Search

RESULTS OF BREADTH-FIRST SEARCH

Results:

Completion time	328	Days
Total Man-Days	1544.77	Man-Days
Total Dev't MD	1332.77	Man-Days
Design & Code	1150.99	Man-Days
QA MD	181.78	Man-Days
Total Testing MD	212.00	Man-Days

	NASA Initial	NASA BREADTH-FIRST Optimal
DEVPRT	0.85	0.8
CUMMD	2093.08	1544.77

D. SENSITIVITY ANALYSIS

Once the optimal distributions for TPFMQA and DEVPRT were found, a series of experiments were conducted to find the sensitivity of these results to changes in various project parameters. The following section will provide a detailed description of the experiments conducted and the results.

1. Experiment 1

The purpose of this experiment was to determine the sensitivity of the optimal QA and testing allocations to changes in the productivity of the workers. In the system dynamics model, the productivity of the workers is disaggregated into two variables, "Nominal Potential Productivity of an Experienced Employee" (NPWPEX) and "Nominal Potential Productivity of a New Employee" (NPWPNE). At any point in time in a project the "Average Nominal Potential Productivity" for the workforce can be found by taking the weighted average of the two parameters [Ref 2:p. 158]. The value for NPWPEX is set in the model to 1 task/man-day and NPWPNE is set to 0.5 tasks/man-day.

The experiment was divided into two sub-experiments. The first was to increase the productivity of both experienced and non-experienced workers by 25% while the second experiment was to decrease productivity by 25%. Figure 4-4 shows the resulting QA distribution for the first experiment. There is a marked increase in the amount of time spent on QA during the coding phase, from 14.6% in the base case to 24.7% in this experiment. This increase can be attributed to the fact that higher productivity not only yields increases in the code generation rate, but also in the error generation rate. Hence more QA is required in the coding phase to discover this higher number of errors. The value for DEVPRT was unchanged by higher productivity.

This can be attributed to the greater effort spent on QA resulting in nearly the same number errors escaping from development to testing.

Figure 4-5 shows the QA distribution resulting from a decrease in workforce productivity. This shows the opposite effect of the previous experiment because lower productivity results in a lower error generation rate, therefore less QA is required. Again DEVPRT was not effected by the change. This can be explained by the fact that the lower QA effort was as effective at detecting errors as the base condition QA due to the lower concentration of errors.

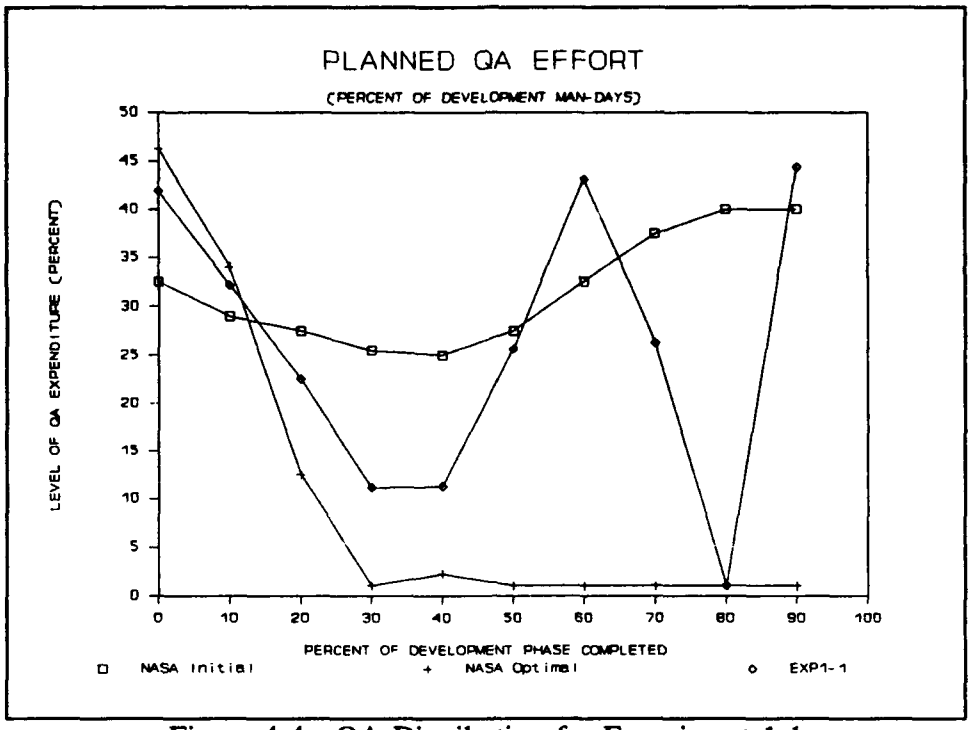

Figure 4-4. QA Distribution for Experiment 1-1

EXPERIMENT 1-1

Purpose: To test the sensitivity of results to changes in the productivity of experienced and inexperienced workers.

Test Results: : Increased the productivity of both new and experienced workers by 25%

Completion time	329	Days
Total Man-Days	1507.64	Man-Days
Total Dev't MD	1347.27	Man-Days
Design & Code	1014.55	Man-Days
QA MD	332.72	Man-Days
Total Testing MD	160.38	Man-Days

	NASA Initial	NASA Optimal	EXP 1-1
DEVPRT	0.85	0.85	0.85
CUMMD	2093.08	1469.79	1507.64

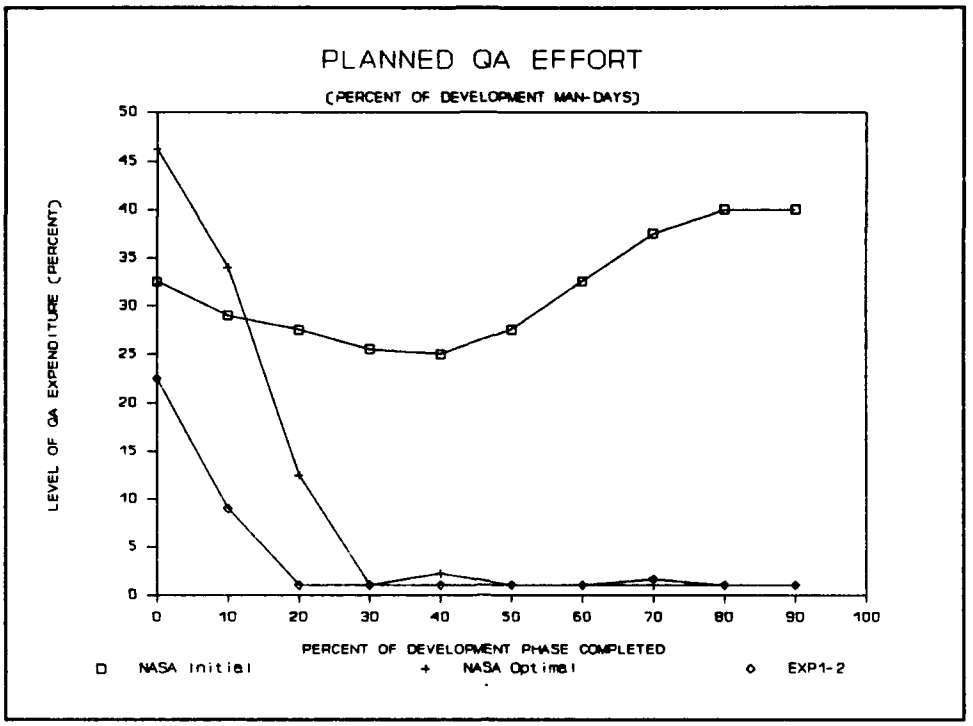

Figure 4-5. QA Distribution for Experiment 1-2

EXPERIMENT 1-2

Purpose: To test the sensitivity of results to changes in the productivity of experienced and inexperienced workers.

Test : Decreased the productivity of both new and experienced workers by 25%
Results:

Completion time	356	Days
Total Man-Days	1718.51	Man-Days
Total Dev't MD	1397.70	Man-Days
Design & Code	1339.29	Man-Days
QA MD	58.40	Man-Days
Total Testing MD	320.81	Man-Days

	NASA Initial	NASA Optimal	EXP 1-2
DEVPRT	0.85	0.85	0.85
CUMMD	2093.08	1469.79	1718.51

2. Experiment 2

The purpose of this experiment was to test the sensitivity of the optimal QA and testing results to changes in the error generation rate. Error generation rate is captured in the model by the variable "Nominal Number of Errors Committed per KDSI" (TNERPK). This variable incorporates all of the various organizational and project-type factors, such as use of structured techniques, quality of staff, project size, etc, that effect the error generation rate. The shape of the curve for TNERPK decreases as the project shifts from design to coding because, as was stated in Chapter II, design errors occur at a higher frequency than coding errors [Ref. 2:p. 193].

This experiment was divided into two sub-experiments. The first was to increase the error generation rate by 50%. Figure 4-6 shows the resulting QA distribution for this experiment. The shape of the curve was somewhat surprising at first because it would seem initially that increasing the number of errors generated should lead to an increase in quality assurance effort. A detailed investigation of the results led to the discovery that the reduction in QA could be attributed to the productivity differences between reworking errors during testing versus reworking them during development.

The amount of effort needed for testing is divided into a fixed component and a variable component. The fixed portion is the effort required to design and execute a test and will be incurred regardless of whether any errors are detected. The variable portion is the effort that must be expended to correct an error once it has been detected. At high error rates the fixed portion becomes negligible so the effort required to detect and rework an error measured in man-days/error will decrease. In this experiment the effort to detect and rework an error during the development phase was found to be on average 0.45 man-days/error while the effort to correct an error during

the testing phase was 0.415 man-days/error[6]. Because of this difference it was more cost effective to defer the majority of the rework effort until the testing phase. This result can also be seen by the decrease in DEVPRT which has the effect of allocating more effort to testing (since testing effort is a function of 1 - DEVPRT).

The second part of this experiment was to decrease the error generation rate by 50%. Figure 4-7 shows the resulting QA distribution for this experiment. The results show the opposite effect of the phenomenon discussed for the previous experiment. In this case the effort required to detect and correct an error during testing was found to be 0.495 man-days/error while during development it remained 0.45 man-days/error. In this case it was more cost effective to rework errors during development so a greater amount of effort is spent on QA than was seen in the base case.

---

[6]The productivity for the testing phase was calcualted by multiply the explosion factor (ratio of errors detected in testing to errors escaped from development) by the nominal effort to rework an error during testing. For this case the numbers were (1661.1/599.99) * 0.15 = 0.415 man-days/error.

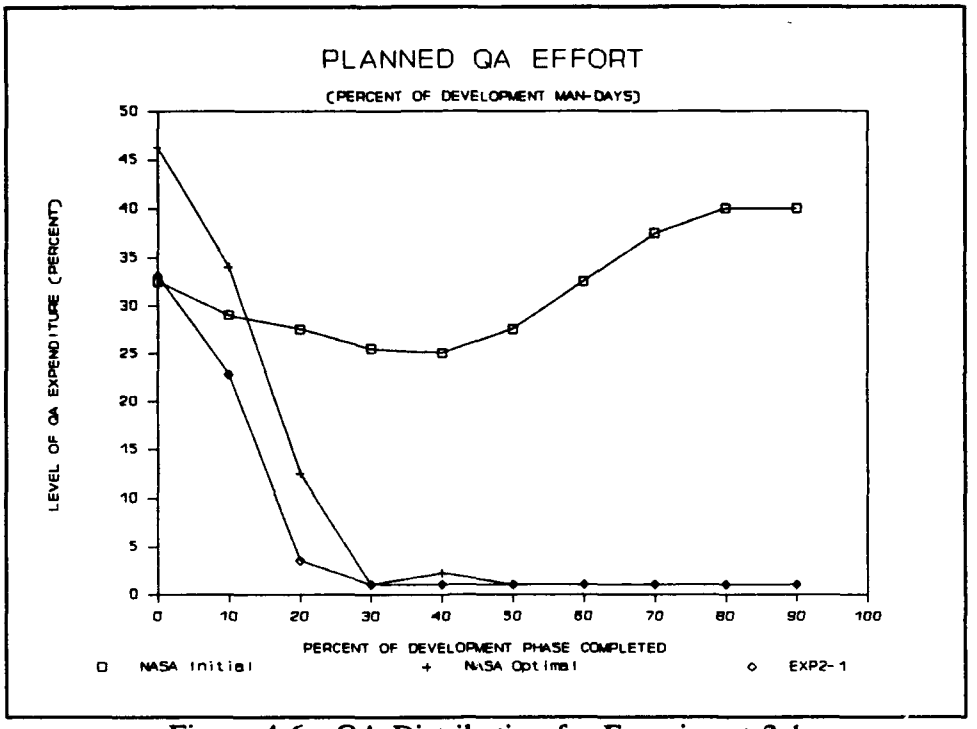

Figure 4-6. QA Distribution for Experiment 2-1

EXPERIMENT 2-1

Purpose: To test the sensitivity of results to changes in the error generation rate.

Test : Increased the nominal number of errors generated per KDSI by 50%.
Results:

Completion time	348	Days
Total Man-Days	1605.02	Man-Days
Total Dev't MD	1217.89	Man-Days
Design & Code	1104.66	Man-Days
QA MD	113.22	Man-Days
Total Testing MD	387.13	Man-Days

	NASA Initial	NASA Optimal	EXP 2-1
DEVPRT	0.85	0.85	0.79375
CUMMD	2093.08	1469.79	1605.02

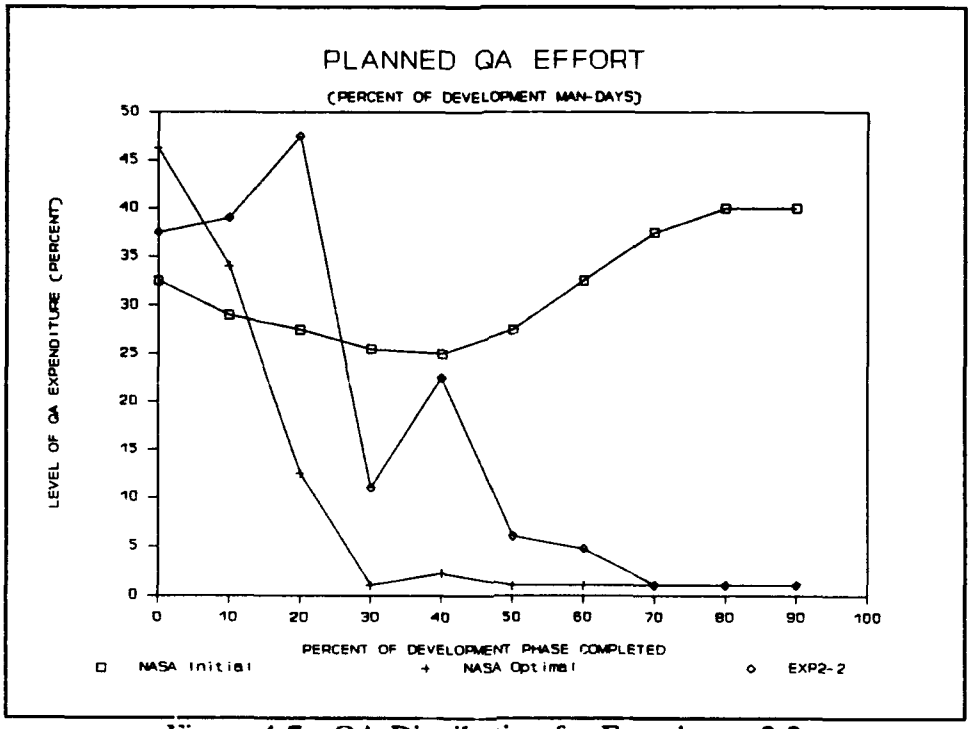

Figure 4-7. QA Distribution for Experiment 2-2

EXPERIMENT 2-2

Purpose: To test the sensitivity of results to changes in the error generation rate.

Test : Decreased the nominal number of errors generated per KDSI by 50%.
Results:

Completion time	331	Days
Total Man-Days	1453.33	Man-Days
Total Dev't MD	1293.93	Man-Days
Design & Code	991.10	Man-Days
QA MD	302.83	Man-Days
Total Testing MD	159.40	Man-Days

	NASA Initial	NASA Optimal	EXP 2-2
DEVPRT	0.85	0.85	0.825
CUMMD	2093.08	1469.79	1453.33

3. Experiment 3

The purpose of this experiment was to determine the sensitivity of the optimal QA and testing results to changes in the amount of rework manpower per error. The effort required to rework an error is captured in the system dynamics model in the variable "Nominal Rework Manpower Needed per Error" (NRWMPE). This value is not a constant, but instead decreases with respect to percent of job completed because design errors in addition to being generated at a higher rate and being more costly to detect, are also more costly to rework [Ref. 2:p. 211].

This experiment also consisted of two sub-experiments. The first was to increase the rework manpower needed per error by 50%. Figure 4-8 shows the resulting QA distribution. The results here are similar to the previous experiment in that the increased cost of reworking errors during development makes it more cost effective to delay rework until the testing phase.

The second experiment was to decrease the rework manpower needed per error by 50%. As was the case in the previous experiment, this has the effect of making reworking errors more cost effective in the development phase so there is a increase in QA effort along with a corresponding decrease in the testing effort.

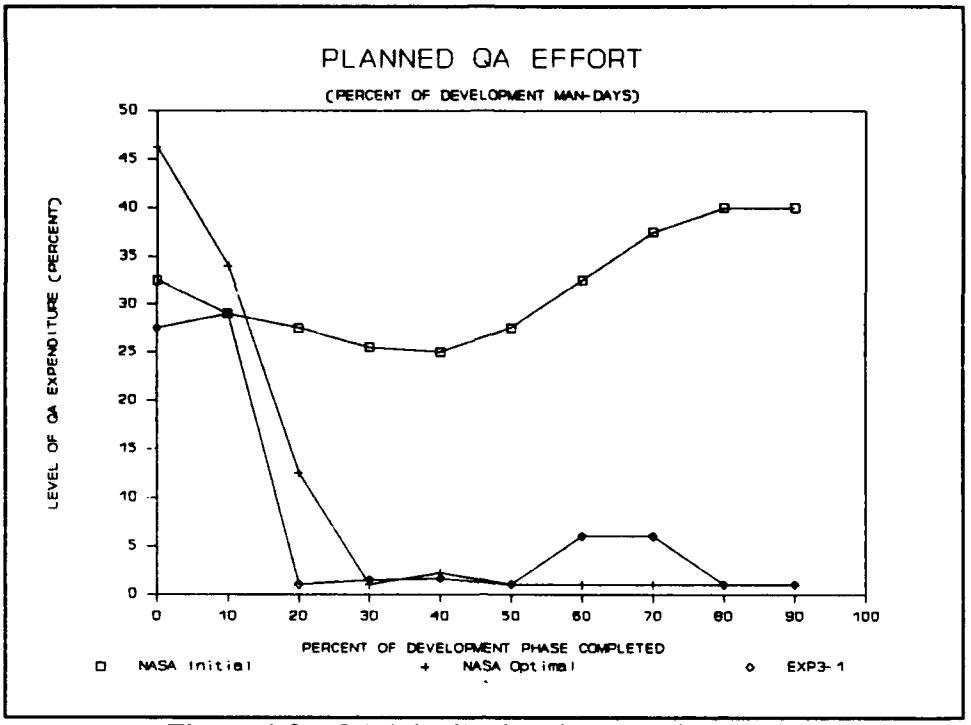

Figure 4-8. QA Distribution for Experiment 3-1

EXPERIMENT 3-1

Purpose: To test the sensitivity of results to changes in the nominal rework manpower per error.

Test : Increased the nominal rework manpower per error by 50%.
Results:

Completion time	331	Days
Total Man-Days	1479.96	Man-Days
Total Dev't MD	1246.75	Man-Days
Design & Code	1122.64	Man-Days
QA MD	124.11	Man-Days
Total Testing MD	233.21	Man-Days

	NASA Initial	NASA Optimal	EXP 3-1
DEVPRT	0.85	0.85	0.8125
CUMMD	2093.08	1469.79	1479.96

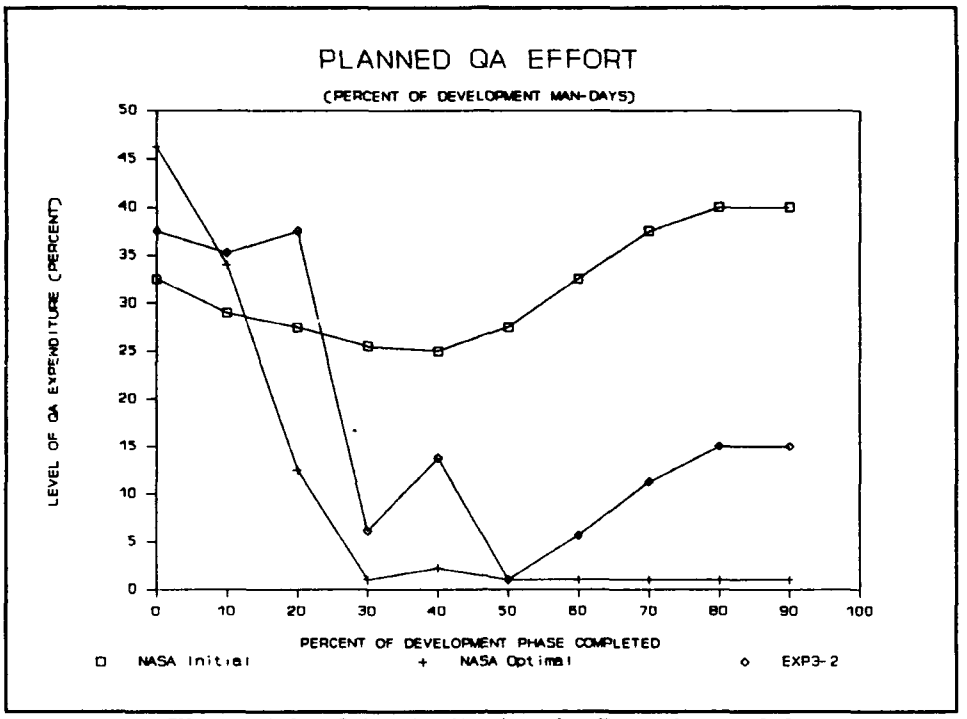

Figure 4-9. QA Distribution for Experiment 3-2

EXPERIMENT 3-2

Purpose: To test the sensitivity of results to changes in the nominal rework manpower per error.

Test Results: : Decreased the nominal rework manpower per error by 50%.

Completion time	331	Days
Total Man-Days	1497.65	Man-Days
Total Dev't MD	1322.94	Man-Days
Design & Code	1062.90	Man-Days
QA MD	260.04	Man-Days
Total Testing MD	174.71	Man-Days

	NASA Initial	NASA Optimal	EXP 3-2
DEVPRT	0.85	0.85	0.85
CUMMD	2093.08	1469.79	1497.65

4. Experiment 4

The purpose of this experiment was to test the sensitivity of the optimal QA and testing distributions to changes in the QA manpower needed to detect the average error. In the model the variable "Nominal QA Manpower Needed to Detect the Average Error" (NQAMPE) is, like NRWMPE, not a constant, but instead decreases with respect to percent of development completed. This is because, as has been reported by several studies, a design error is as much as 3 times more costly to detect and correct than a coding error [Ref. 2:p. 205].

The first sub-experiment in this case was to increase the effort needed to detect an error during QA by 50%. Figure 4-10 shows the resulting QA distribution. For this case there is very little change in the distribution of effort for QA, however, since fewer errors are being detected by QA more effort is required in testing as is demonstrated by the decrease in DEVPRT.

For the second sub-experiment the effort to detect an error during QA was decreased by 50%. Figure 4-11 shows the resulting QA distribution for this sub-experiment. The decrease in quality assurance effort during design can be attributed to the fact that since it takes less effort to detect an error the same number of errors can be detected with a lesser effort. In the coding phase we see an increase in the QA effort because those errors that were being delayed until testing in the base case are now more cost effective to detect during the development phase. Accordingly there is a decrease in the amount of effort used in testing because of the lower error density being passed into the testing phase.

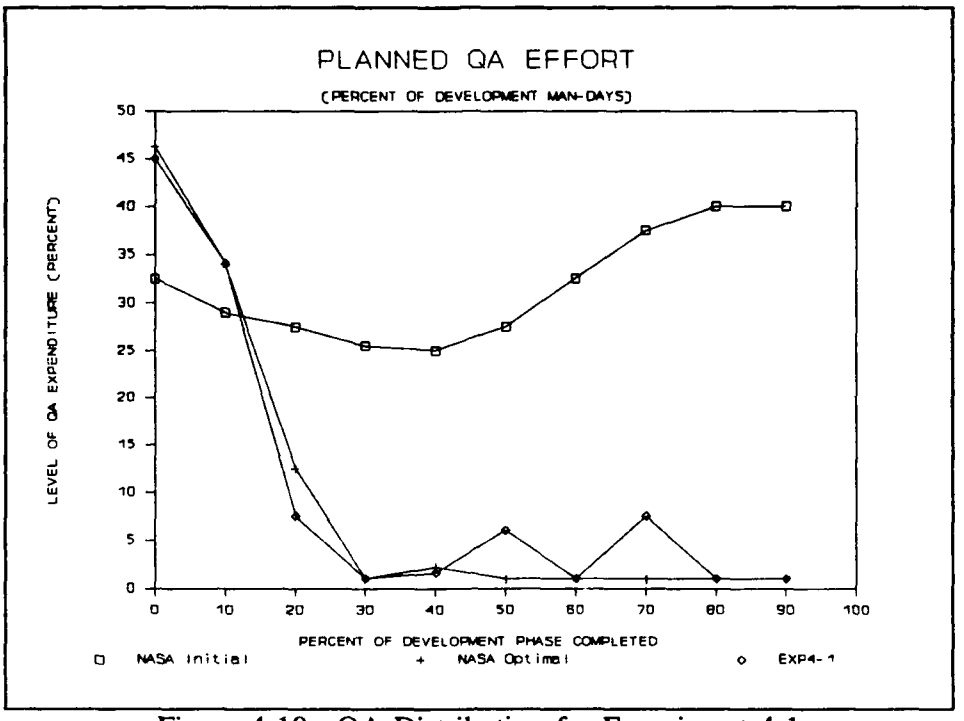

Figure 4-10. QA Distribution for Experiment 4-1

EXPERIMENT 4-1

Purpose: To test the sensitivity of results to changes in the nominal QA manpower needed to detect the average error.

Test : Increased the nominal QA manpower needed to detect the average error by 50%.

Results:

Completion time	331	Days
Total Man-Days	1473.74	Man-Days
Total Dev't MD	1249.85	Man-Days
Design & Code	1069.50	Man-Days
QA MD	180.36	Man-Days
Total Testing MD	223.88	Man-Days

	NASA Initial	NASA Optimal	EXP 4-1
DEVPRT	0.85	0.85	0.81875
CUMMD	2093.08	1469.79	1473.74

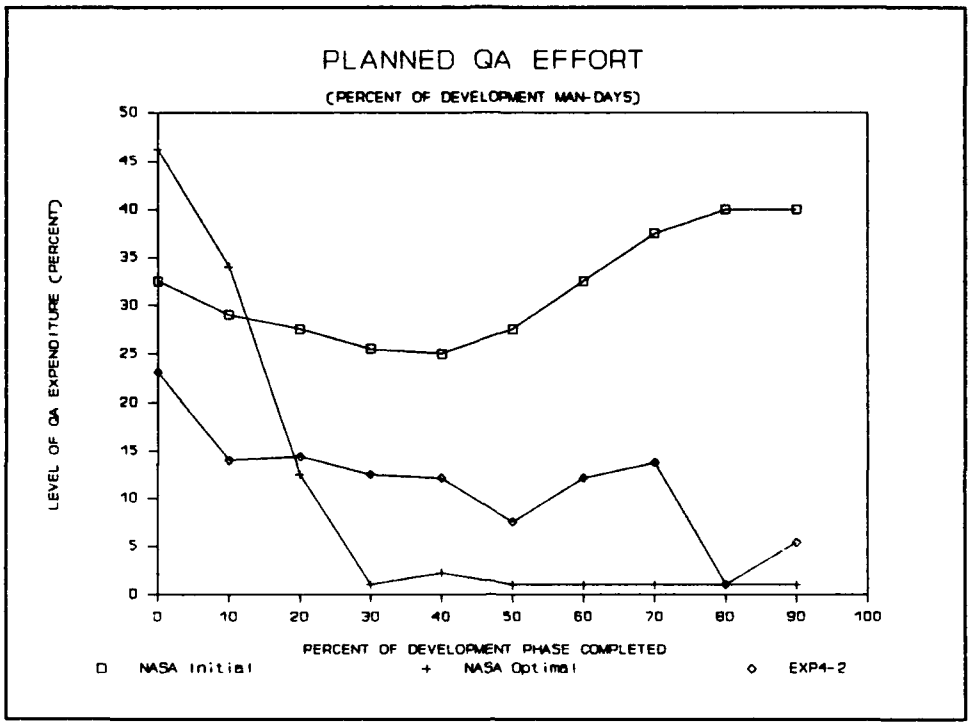

Figure 4-11. QA Distribution for Experiment 4-2

EXPERIMENT 4-2

Purpose: To test the sensitivity of results to changes in the nominal QA manpower needed to detect the average error.

Test : Decreased the nominal QA manpower needed to detect the average error by 50%.

Results:

Completion time	329	Days
Total Man-Days	1522.36	Man-Days
Total Dev't MD	1362.03	Man-Days
Design & Code	1202.20	Man-Days
QA MD	159.83	Man-Days
Total Testing MD	160.32	Man-Days

	NASA Initial	NASA Optimal	EXP 4-2
DEVPRT	0.85	0.85	0.85
CUMMD	2093.08	1469.79	1522.36

5. Experiment 5

The purpose of this experiment was to test the sensitivity of the optimal QA and testing policies to changes in the percent of bad fixes. "Percent of Bad Fixes" (PBADFX) is a constant variable in the model used to express the percentage of error fixes done during rework that are incorrect and as such result in new errors. Bad fixes are generated in various ways, such as corrections based on faulty analysis or by the correction resulting in the creation of a new error [Ref. 2:p. 214]. In the model the value for PBADFX is set to 0.075.

The first sub-experiment was to increase PBADFX to 0.15. Figure 4-12 shows the resulting QA distribution for this experiment. There was a slight decrease in the amount of QA done during the design phase because bad fixes introduced here will multiply to become coding and documentation errors that will result in an increase in the amount of effort required for testing.

The second experiment was to decrease PBADFX to 0.05. Figure 4-13 shows the resulting QA distribution. The curve is very similar to the base case with the exception of a large spike in the middle of the coding phase. A detailed investigation of the results failed to find a plausible reason for this result.

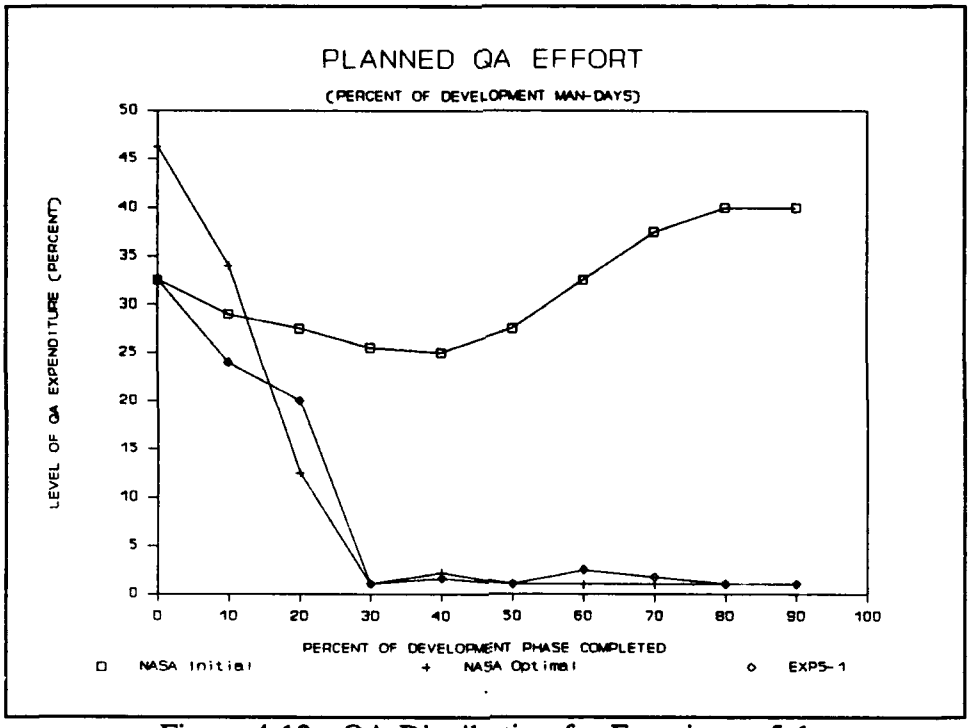

Figure 4-12. QA Distribution for Experiment 5-1

EXPERIMENT 5-1

Purpose: To test the sensitivity of results to changes in the percent of bad fixes.

Test : Increased the percent of bad fixes by 50%.

Results:

Completion time	332	Days
Total Man-Days	1504.76	Man-Days
Total Dev't MD	1254.95	Man-Days
Design & Code	1100.96	Man-Days
QA MD	153.99	Man-Days
Total Testing MD	249.81	Man-Days

	NASA Initial	NASA Optimal	EXP 5-1
DEVPRT	0.85	0.85	0.8
CUMMD	2093.08	1469.79	1504.76

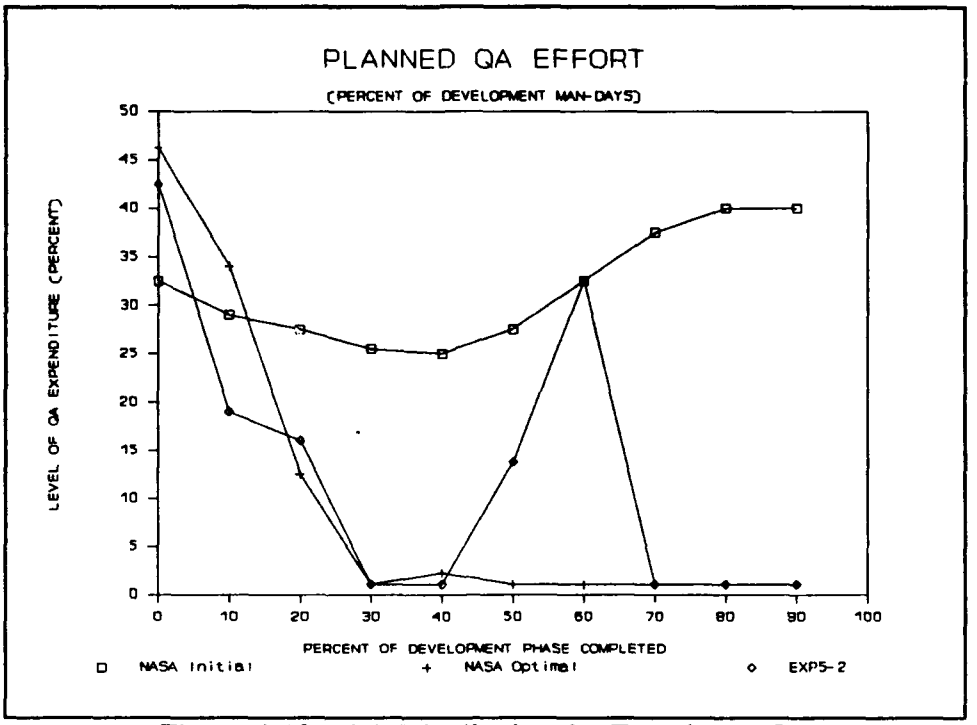

Figure 4-13. QA Distribution for Experiment 5-2

EXPERIMENT 5-2

Purpose: To test the sensitivity of results to changes in the percent of bad fixes.

Test : Decreased the percent of bad fixes from 7.5% to 5%.

Results:

Completion time	333	Days
Total Man-Days	1506.38	Man-Days
Total Dev't MD	1278.94	Man-Days
Design & Code	1008.63	Man-Days
QA MD	190.31	Man-Days
Total Testing MD	227.45	Man-Days

	NASA Initial	NASA Optimal	EXP 5-2
DEVPRT	0.85	0.85	0.79375
CUMMD	2093.08	1469.79	1506.38

6. Experiment 6

The purpose of this final experiment was to determine the sensitivity of the optimal QA and testing policies to errors (or the lack there of) in estimation of project size and schedule. As a software project develops, project managers often realize that they have under-estimated the number of tasks that comprise the software system being developed [Ref. 2:p. 249]. The variable "Tasks Underestimation Fraction" (UNDEST) is used in the model to simulate this under-estimation. As an example of its use would consider a project whose actual size is 100 tasks and UNDEST is set to 0.25. The "Perceived Job Size in Tasks" at the beginning of the project would be 100 * 0.25 = 75 tasks.

This experiment consisted of only one part, to simulate perfect estimation by setting UNDEST to 0 and making the initial estimates for job size and completion time equal to the actual values found in the project. Figure 4-14 shows the resulting QA distribution. The pattern of this curve is essentially the same as the base case but the values are lower for the design phase. This can be attributed to the fact that since the job size is accurately known at the beginning of the project, the staff size is larger during the early stages. Therefore, the percentage of effort may be lower for this experiment, but it is a lower percentage of a higher absolute number so the actual effort expended during this phase is nearly equal to the base case.

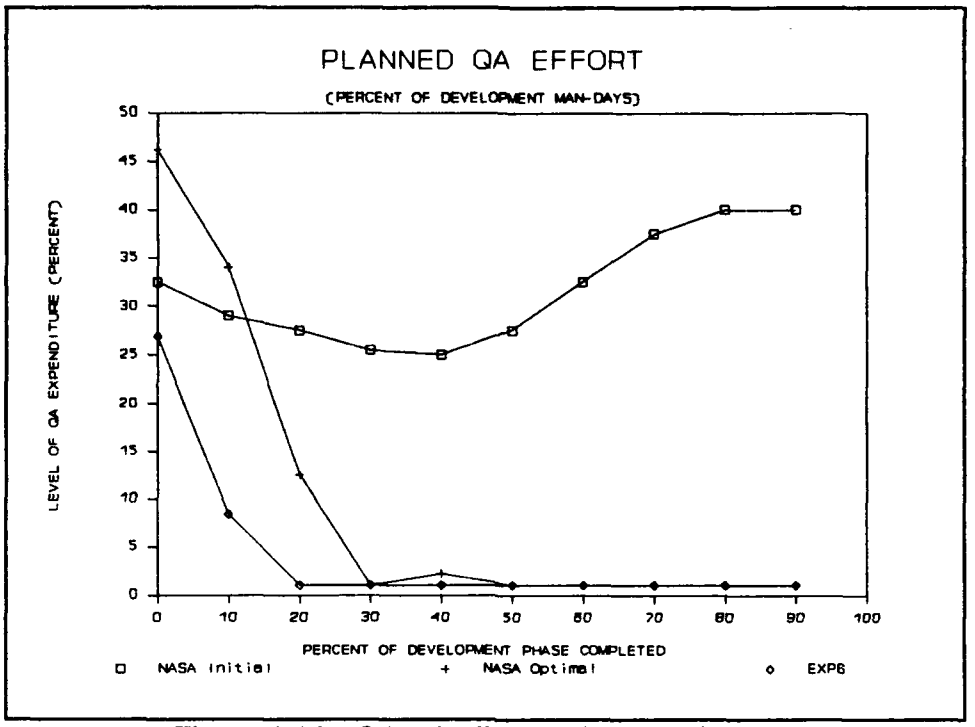

Figure 4-14. QA Distribution for Experiment 6

EXPERIMENT 6

Purpose: To test the sensitivity of results to changes in the accuracy of initial project estimation for size and time.

Test : Changed the initial estimates for project size and completion time to the actual size and time to simulate perfect estimation

Results:

Completion time	396	Days
Total Man-Days	1455.10	Man-Days
Total Dev't MD	1221.63	Man-Days
Design & Code	1167.68	Man-Days
QA MD	53.95	Man-Days
Total Testing MD	233.47	Man-Days

	NASA Initial	NASA Optimal	EXP 6
DEVPRT	0.85	0.85	0.975
CUMMD	2093.08	1469.79	1455.10

V. CONCLUSIONS AND TOPICS FOR FUTURE RESEARCH

A. CONCLUSIONS

The primary objective of this thesis was the development of an expert system to simultaneously find the optimal value for two variables using the System Dynamics Model for Software Project Management. As a first step two expert system modules were designed, the first to optimize a single valued variable, the second to optimize a table function. Using these two modules, two other expert system modules were designed to accomplish the two variable optimization using depth-first and breadth-first search techniques.

The NASA DE-A software project was used as a test case for both modules. The objective was to find optimal allocations for QA and testing (the variables TPFMQA and DEVPRT respectively). While both programs found QA and testing allocations that resulted in lower project costs, the depth-first search algorithm proved to be more efficient and realized the lowest cost. A series of tests were then done to find the sensitivity of the results to changes in the following:

1. Productivity of the workers
2. Error generation rates
3. Effort to rework an error during development
4. QA manpower needed to detect an error
5. Percentage of bad fixes
6. Estimation error for project size and schedule.

From these tests it was determined that the results are most sensitive to changes in productivity and error generation rate.

B. TOPICS FOR FUTURE RESEARCH

Several areas are available for conducting future research. Two prominent topics are: (1) refine the current expert system modules, or (2) conduct experiments using the current modules to optimize other variable pairs.

1. Refining the Current Expert Systems Modules

Since the purpose of this thesis was to determine the feasibility of optimizing two variables the depth-first and breadth-first search techniques were chosen because they were the simplest to implement. A possible follow-on project would be to investigate other search techniques such as those presented in Chapter I.

Another possible follow-on topic would be to develop and expert system module that uses knowledge of a particular application (in this case allocation of manpower to QA and testing) to make changes as the project is being simulated by using the gaming interface. The current system makes adjustments to the variables after the entire project is complete and then reruns the simulation to determine the effects of the changes. The new system would simulate an expert project manager controlling a software project in real time. The expert system should "learn" from each successive project simulation to further refine the rule base.

2. Use the Expert System Modules to Conduct Further Experiments

This thesis concentrated on finding the optimal allocation of manpower to QA and testing. However, the modules were designed to optimize any pair of variables (where one is single valued and one is a table function) that the user may chose. A series experiments could be conducted to find the optimal values for other related variable pairs in the model.

APPENDIX A

```
/*************************************************************************
 *  PROGRAM : SINGLE.ARI
 *  AUTHOR  : LT C E AGAN
 *  ADVISOR : PROFESSOR T. K. ABDEL-HAMID
 *  DATE    : 1/3/90
 *  REVISION: 2/4/90
 *  SYSTEM  : IBM compatible PC using ARITY Prolog Interpreter
 *  PURPOSE : This program is designed to be used with the DYNAMO Model
 *  for Software Project Management to find the minimum cost, measured in
 *  cumulative man-days (the variable CUMMD in the DYNAMO Model), for a
*  given project by adjusting the value of a single constant variable. The name of
 *  the variable along with initial value and pulse size are supplied by the
 *  user. A description of the algorithm used for optimization is given later
 *  in the code.
 *************************************************************************/

/*************************************************************************
 *                    PROGRAM REQUIREMENTS
 *
 *      This program was written for use with an Arity Prolog Interpreter.
 *  Comments are included in the code where non-standard (Clocksin-Mellish)
 *  predicates were used. These code segments may requ  changes before
 *  running on a different prolog system.
 *      The program requires that the following files be in the same
 *  directory as the program when it is executed:
 *      1. PROJECT.DYN - The dynamo file containing the program for the
 *         project to be simulated (and all the other files with the names
 *         project.* that are required by dynamo to run a simulation).
 *      2. PROJECT.DRS - The Dynamo report format file that is used to
 *         write the result of the simulation to the file PROJECT.OUT for
 *         the program to read (PROJECT.OUT will be created and need not
 *         exist prior to running the program).
 *      3. MODEL.BAT - A batch file containing the DOS commands to simulate
 *         the project using DYNEX and to output the results (This file calls
 *         the following EXE files that must also be in the default directory:
 *         DYNEX.EXE, SMLT.EXE, REP.EXE. These three files are all part of
 *         the Professional Dynamo Simulator Version 3.3).
 *************************************************************************/

/*************************************************************************
 *                    INSTRUCTIONS
 *
 *    To run this program first initiate the Arity Prolog Interpreter by
 *    typing API at the DOS prompt. When the prolog prompt (?-) appears
 *    type [single]. (including the period) and press return. The
 *    interpreter should then answer with yes and return to ?- . At this
```

```
*     point type optimize_single. and press return. The program will then
*     ask a series of questions to be answered by the user (such as the
*     name of the variable to be adjusted). Enter your reply and press
*     enter (NOTE -- you need not follow the answer with a period).
*************************************************************************/

/*************************** SOURCE CODE ****************************/


/* Rule optimize_single is the main module for the single valued variable
 * optimization module. It first calls the rule init_vars to get the initial
 * values for the variables and then calls the rule minimize for the first
 * starting point. Once it returns the results from the first value are
 * printed and then destroyed and minimize is called with the second starting
 * point. Two points, one high one low are used to overcome the problem of
 * a bimodal distribution.
 */

optimize_single :-
   init_vars(Pulse,Value1,Value2),

/* Test at the initial point      */

   get_cummd(Value1,0,Cummd),     /* Find CUMMD at first point */
   minimize(Value1,Pulse,0),
   output_message('The first minimum is at '),

/* Test at the second point  */

   abolish(result/3),             /* Destroy results from the first point */
   abolish(size/1),

   asserta(size(Pulse)),          /* Return to the initial pulse size */

   get_cummd(Value2,0,NewCummd), /* Find CUMMD at second point */

   minimize(Value2,Pulse,0),
   output_message('The second minimum is at ').

/* Rule minimize finds the minimum value of the function by using the
 * following algorithm:
 *   1. See if the exit condition has been met. For this program the exit
 *      condition is when the pulse size is reduced below the user
 *      supplied minimum pulse size. When the exit condition is met the
 *      rule is finished.
 *   2. Find the direction of improvement by first applying a negative pulse
 *      and comparing the resulting value of CUMMD with the original value.
 *      If the new value is greater then apply a positive pulse and perform
 *      the same test. If neither pulse is better then the pulse size is
 *      halved and return to step 1.
 *   3. If a direction of improvement is found then continue moving in that
 *      direction until improvement is no longer noted. The last value
```

```
 *       where improvement was found becomes the new minimum.
 *    4. Halve the pulse size and return to step 1 with the minimum point
 *       as the new starting point.
 */

/* If the exit condition is met end the rule  */

minimize(Value,Pulse,I) :-
    exit_condition, !.

/*  Otherwise continue to execute the search algorithm  */

minimize(Value,Pulse,I) :-
      minval(Minval),
      maxval(Maxval),
      NewI is I + 1,

      /* Pulse in the negative direction  */
      NewValue is Value - Pulse,

      /* Ensure that the new value is not less then the minimum */
      range_check(NewValue,Maxval,Minval,ScaledVal),

      /* Retrieve the old result and run the model to find the new result */
      result(I,Value,OldC),
      get_cummd(ScaledVal,NewI,NewC),

      /* If the negative pulse was an improvement then set the value of Pulse
       * to Pulse * -1, otherwise pulse in the positive direction
       * NOTE-- the ifthenelse predicate is not standard prolog
       */
      ifthenelse(NewC =< OldC,pulse_neg(Pulse),
                              pulse_pos(Value,Pulse,NewI)),

      /* Continue in the direction of improvement starting at the last
       * point tested by calling the rule find_min
       */
      size(NewPulse),
      result(NewI1,NewValue1,NewC1),
      find_min(NewValue1,NewPulse,NewI1,NewC1,OldC),

      /* Retract the result from the point where improvement was no longer
       * noted */
      retract(result(A,B,C)),

      /* Retrieve the result from the last point of improvement and assert it
       * as the new minimum
       */
      result(MinI,MinValue,MinCummd),
      asserta(min(MinValue,MinCummd)),

      /* Halve the Pulse size and return to the beginning of the rule */
```

```
    SmallerPulse is Pulse * 0.5,
    abolish(size/1),
    asserta(size(SmallerPulse)),
    minimize(MinValue,SmallerPulse,MinI).


/* Rule pulse_neg is called when an initial negative pulse yields improvement
 * over the previous value. Since the value of pulse is always added to the
 * previous value of the variable, this rule changes the sign of Pulse to
 * negative in order to have the effect of subtraction.
 */

pulse_neg(Pulse) :-
    NewPulse is Pulse * -1,
    abolish(size/1),
    asserta(size(NewPulse)).

/* Rule pulse_pos is called whenever a negative pulse did not yield
 * improvement. First the result from the negative pulse is retracted,
 * then a positive pulse is added to the original value. Rule get_cummd
 * is then called to find the value of CUMMD at the new point.
 */

pulse_pos(Value,Pulse,I) :-
    minval(Minval),
    maxval(Maxval),
    retract(result(I,V,X)),
    NewValue is Value + Pulse,
    range_check(NewValue,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,I,Cummd).


/* Rule find_min performs the iteration that continues to pulse in the
 * given direction until it no longer leads to improvement in CUMMD.
 */

/* If the new CUMMD is more than the old CUMMD then exit the rule */
find_min(Value,Pulse,I,NewC,OldC) :-
    NewC > OldC,!.

/* If the current pulse is positive and the variable is at the maximum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    maxval(Maxval),
    Pulse > 0,
    Value == Maxval,!.

/* If the current pulse is negative and the variable is at the minimum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
```

```
    minval(Minval),
    Pulse < 0,
    Value == Minval,!.

/* Otherwise pulse in the current direction of the variable Pulse and
 * then run the model to get the new value of CUMMD.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    minval(Minval),
    maxval(Maxval),

    /* Pulse in the previously set direction of improvement */
    NewVal is Value + Pulse,

    /* Ensure that the new value is between the user supplied minimum
     * and maximum value
     */
    range_check(NewVal,Maxval,Minval,ScaledVal),

    /* Get the new value of CUMMD and iterate the rule through the
     * use of tail recursion.
     */
    NewI is I + 1,
    get_cummd(ScaledVal,NewI,Cummd),
    find_min(ScaledVal,Pulse,NewI,Cummd,NewC).

/* Rule get_cummd first tests to see if a result already exists for the
 * given value. If not the model is run to find the value for CUMMD at
 * the given value for the test variable.
 */

/* If a result already exists for Value then exit the rule. */
get_cummd(Value,I,Cummd) :-
    result(I,Value,Cummd).

/* Otherwise, run the model to find the value of CUMMD at the given point */
get_cummd(Value,I,Cummd) :-
    output_value(Value),          /* write the value to project.dnx */
    shell(model),                 /* run the model */
    read_cummd(Cummd),            /* read the result from project.out */
    output_cummd(Cummd,I),        /* write the result to summary.dat */
    asserta(result(I,Value,Cummd)). /* add the result to the database */

/* Rule exit_condition succeeds if the size of Pulse is less than the
 * minimum pulse size given by the user.
 */

exit_condition :-
    size(Pulse),
    minsize(MinPulse),!,
    Pulse < MinPulse.
```

```
/* Rule output_value writes the name and value of the test variable to the
 * files project.dnx and summary.dat
 */

output_value(Value) :-
    var(VarName),      /* retrieve the name of the variable */

    /* write the name and value to project.dnx so they can be used by
     * the model in executing the simulation.
     * /
    create(D,'project.dnx'),
    write(D,'C '), write(D,VarName),write(D,'='),
    write(D,Value),nl(D),
    close(D),

    /* write the name and value to summary.dat for later analysis by the
     * user.
     */
    open(S,'summary.dat',a),
    write(S,VarName), write(S,'= '),
    write(S,Value),nl(S),
    close(S).

/* Rule read_cummd reads the value of CUMMD from the file project.out
 * after the simulation is complete.
 */

read_cummd(Cummd) :-
    open(C,'project.out',r),
    read(C,Output),
    asserta(Output),
    cummd(Cummd),
    close(C).

/* Rule output_cummd writes the values of CUMMD and ITER passed to it
 * to the file summary.dat.
 */

output_cummd(Cummd,Iter) :-
    open(S,'summary.dat',a),
    write(S,Iter),write(S,'. '),
    write(S,'CUMMD= '),
    write(S,Cummd),nl(S),
    close(S).

/* Rule output_message writes the given message and the current minimum to
 * the file summary.dat. The model is then run with the final value so the
 * results in the file project.rsl will reflect the optimal values.
 */

output_message(Message) :-
```

```
    min(MinValue,MinCummd),
    open(S,'summary.dat',a),
    nl(S),
    write(S,'*****************************************'),
    nl(S),
    write(S,Message),
    write(S,MinValue),
    write(S,' with cummd '),
    write(S,MinCummd),
    nl(S),
    close(S),

    /* Run the model with the optimal value.  */
    get_cummd(MinValue,0,MinCummd).

/* Rule init_vars prompts the user for the initial values used in
 * the program and stores the values in the database.
 */

init_vars(Pulse,Value1,Value2) :-
    asserta(cycle(0)),
    write('What variable do you want to optimize? '),
    /* NOTE-- the predicate atom string is not standard prolog.  It is
     * used to read a character string of length n (the first argument)
     * from the screen (standard input).
     */
    read_string(30,ArString),nl,

    /* NOTE-- the predicate atom_string is not standard prolog. It is used
     * to convert a string variable (the second argument) to a standard
     * prolog atom (the first argument).
     */
    atom_string(VarName,ArString),

    asserta(var(VarName)),
    write('What is the maximum value for the variable? '),
    readnumber(Maxval),
    asserta(maxval(Maxval)),
    write('What is the minimum value for the variable? '),
    readnumber(Minval),
    asserta(minval(Minval)),
    write('What is your desired pulse size? '),
    readnumber(Pulse),
    asserta(size(Pulse)),
    write('What is your minimum pulse size? '),
    readnumber(MinPulse),
    asserta(minsize(MinPulse)),
    write('What is the value for the first test point? '),
    readnumber(Value1),
    asserta(test(Value1)),
    write('What is the value for the second test point? '),
    readnumber(Value2),
```

```
    asserta(test(Value2)),
    output_header(VarName,Minval,Maxval,Pulse,MinPulse).

/* Rule output_header writes the initial values for the given variables to
 * the file summary.dat.
 */

output_header(VarName,Minval,Maxval,Pulse,MinPulse) :-
    create(S,'summary.dat'),
    write(S,'       Variable Name      = '),write(S,VarName),nl(S),
    write(S,'       Maximum Value       = '),write(S,Maxval),nl(S),
    write(S,'       Minimum Value       = '),write(S,Minval),nl(S),
    write(S,'       Pulse size factor  = '),write(S,Pulse),nl(S),
    write(S,'       Minimum Pulse Size = '),write(S,MinPulse),nl(S),
    nl(S),
    close(S).

/* Rule readnumber reads the user input value as a string and then
 * converts it to a number.  This routine is used to avoid forcing
 * the user to type a period at the end of each entry.  It makes use of
 * the arity built-in predicates read_string and float_text which are not
 * standard prolog predicates.
 */

readnumber(Result) :-
    read_string(255,ArString),nl,
    float_text(Result,ArString,_).

/* If the first rule fails because the user did not type a number then
 * make him try again.
 */
readnumber(Result) :-
    write('Number expected, try again'),
    nl,
    readnumber(Result).

/* Rule range_check ensures that the input variable is between the
 * minimum and maximum value supplied by the user and if not it assigns
 * the value of the minimum or maximum value to the result
 */
/* If the given value is greater than the maximum return the value of
 * the maximum.
 */
range_check(Val,Max,Min,Result) :-
    Val > Max,
    Result is Max,!.

/* If the given value is less than the minimum return the value of the
 * minimum.
 */
range_check(Val,Max,Min,Result) :-
    Val < Min,
```

```
    Result is Min,!.

/* Otherwise return the given value  */
range_check(Val,Max,Min,Result) :-
    Result is Val,!.
```

APPENDIX B

```
/*************************************************************************
 *  PROGRAM : TABLE.ARI
 *  AUTHOR  : LT C E AGAN
 *  ADVISOR : PROFESSOR T. K. ABDEL-HAMID
 *  DATE    : 12/20/89
 *  REVISION: 2/5/90
 *  SYSTEM  : IBM compatible PC using ARITY Prolog Interpreter
 *  PURPOSE : This program is designed to be used with the DYNAMO Model
 *  for Software Project Management to find the minimum cost, measured in
 *  cumulative man-days (the variable CUMMD in the DYNAMO Model), for a
 *  given project by adjusting the value of a single table variable. The name of
 *  the variable along with initial values and pulse size are supplied by the
 *  user.  A description of the algorithm used for optimization is given later
 *  in the code.
*************************************************************************/


/*************************************************************************
 *                    PROGRAM REQUIREMENTS
 *
 *      This program was written for use with an Arity Prolog Interpreter.
 *  Comments are included in the code where non-standard (Clocksin-Mellish)
 *  predicates were used.  These code segments may require changes before
 *  running under a different prolog system.
 *      The program requires that the following files be in the same
 *  directory as the program when it is executed:
 *      1. PROJECT.DYN - The dynamo file containing the program for the
 *         project to be simulated (and all the other files with the names
 *         project.* that are required by dynamo to run a simulation).
 *      2. PROJECT.DRS - The Dynamo report format file that is used to
 *         write the result of the simulation to the file PROJECT.OUT for
 *         the program to read (PROJECT.OUT will be created and need not
 *         exist prior to running the program).
 *      3. MODEL.BAT - A batch file containing the DOS commands to simulate
 *         the project using DYNEX and to output the results (This file calls
 *         the following EXE files that must also be in the default directory:
 *         DYNEX.EXE, SMLT.EXE, REP.EXE.  These three files are all part of
 *         the Professional Dynamo Simulator Version 3.3).
 *************************************************************************/

/*************************************************************************
 *                    INSTRUCTIONS
 *
 *     To run this program first initiate the Arity Prolog Interpreter by
 *     typing API at the DOS prompt.  When the prolog prompt (?-) appears
 *     type [table]. (including the period) and press enter.  The
```

```
*      interpreter should then answer with "yes" and return to ?- . At this
*      point type optimize_table. and press enter. The program will then
*      ask a series of questions to be answered by the user (such as the
*      name of the variable to be adjusted). Enter your reply and press
*      enter (NOTE -- you need not follow the answer with a period).
*************************************************************************/

/*************************** SOURCE CODE ****************************/

/* Rule optimize_table is the main module for the table function pattern
 * search optimization module. It first queries the user for the required
 * input and then runs the simulation for the initial points. The rule
 * pattern_search is then called to execute the pattern search algorithm.
 */

optimize_table :-
    /* Ask the user for the necessary input.
     * NOTE-- the predicates read_string and atom_string are built in
     * Arity predicates and are not standard prolog.
     */
    write('What table variable do you want to optimize? '),
    read_string(30,String_VarName),nl,
    atom_string(VarName,String_VarName),
    asserta(var(VarName)),
    write('How many values are in your table? '),
    readinteger(TableSize),
    asserta(tabsize(TableSize)),
    write('What is the maximum value for a variable in the table? '),
    readnumber(Maxval),
    asserta(max(Maxval)),
    write('What is the minimum value for a variable in the table? '),
    readnumber(Minval),
    asserta(min(Minval)),
    write('What is your desired pulse size? '),
    readnumber(Pulse),
    asserta(size(Pulse)),
    write('What is the minimum pulse size? '),
    readnumber(MinPulse),
    asserta(minsize(MinPulse)),

    /* Write the users supplied values to the file summary.dat  */
    output_header(VarName,TableSize,Maxval,Minval,Pulse,MinPulse),

    /* Get the starting values for the table  */
    foriterate(get_initial_values,TableSize),

    /* Execute the model with the starting values and then call the rule
     * pattern_search to start the algorithm.
     */
    initial_run,
    pattern_search.
```

```
/* Rule pattern_search executes the pattern search algorithm (Hooke and
 * Jeeves - 1961). The basic theory of the algorithm is as follows:
 *    1. Begin at an arbitrary point base1 (user supplied initial points).
 *    2. Perturb each of the variables in the table by first adding and then
 *       subtracting the pulse size from the value. If either pulse results
 *       in a value for CUMMD less than the base then it becomes the new
 *       temporary head. Otherwise the variable is returned to its original
 *       value.
 *    3. When all the perturbations are complete if value for CUMMD at the
 *       final temporary head is less than at base1 then it becomes the new
 *       base2. Since the previous move resulted in improvement, the
 *       algorithm assumes that another move in the same direction will also
 *       yield improvement so the pattern is jumped to a new temporary head.
 *       The jump is accomplished by adding to each of the values in base2
 *       the difference between the base2 and base1 values for that point.
 *       Finally base2 becomes the new base1 and return to step 2.
 *    4. If the final temporary head is not an improvement over base1 then
 *       the pattern retreats to base1 to do local perturbation around it
 *       to try to establish a new pattern. If a new better temporary head
 *       is found then the pattern is jumped and return to step 2.
 *    5. If local explorations around the base do not result in a new pattern
 *       then the previous pattern is destroyed, the pulse size is halved,
 *       and the base1 values become the starting point for a new pattern.
 *    6. The search terminates when the pulse size falls below the user set
 *       minimum pulse size.
 */


/* Test if the exit condition is true  */
pattern_search :-
    exit_condition.

/* Case where final temporary head is better than previous base so temporary
 * head becomes the new base2 and the pattern is jumped.
 */

pattern_search :-
    /* NOTE-- the snip operators ([!,!]) are not standard prolog.  Their
     * purpose is similar to the cut operator (!) except that they only
     * prevent backtracking into the statements inside the snips and they do
     * not have the side effect of preventing execution of other predicates
     * of the same name as the one where they were executed.
     */

    /* Retrieve the value for CUMMD at the current base and the size of the
     * table from the database.
     */
    [!basecummd(OldCummd),
    tabsize(TableSize),
```

```
    /* Perturb each of the variables in the table  */
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store the values for each variable to the list temp */
    result(NewCummd)!], /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd),

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only be executed if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve the current values for the temp head */

    /* Set base2 equal to the temporary head and set the value basecummd
     * to NewCummd.
     */
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
    asserta(basecummd(NewCummd)),

    /* Jump the pattern to the new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.


/* Case where temp head is not better than base so try local exploration
 * about the base.

pattern_search :-

    /* First test to see if already at base1.  If not retreat to base1 and
     * perform local explorations.
     */
    [!not(already_at_base1),
    output_message('Retreat to Base for Local Exploration'),
    retrieve_base1_values,

    /* Retrieve the table size and value for CUMMD at the base from the
     * database.
     */
    tabsize(TableSize),
    basecummd(OldCummd),

    /* Perturb each of the variables in the table  */
```

```
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store the values for each variable to the list temp */
    result(NewCummd)!], /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd),

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only be executed if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve the current values for the temp head */

    /* Set base2 equal to the temporary head and set the value basecummd
     * to NewCummd.
     */
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
    asserta(basecummd(NewCummd)),

    /* Jump the pattern to the new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.

/* Case where local explorations about base1 do not result in improvement so
 * reduce pulse size and start pattern over from base1.
 */

pattern_search :-
    output_message('Reduce Pulse Size'), /* Annotate summary.dat  */
    retrieve_base1_values,

    /* Reduce pulse size and start pattern search over  */
    size(Pulse),
    NewPulse is Pulse * 0.5,
    retract(size(Pulse)),
    asserta(size(NewPulse)),
    pattern_search.

/* Rule perturb_variable is passed to foriterate and is used to retrieve the
 * necessary variables and then call do_perturbation in order to perturb
 * each value in the table.
 */

perturb_variables :-
```

```
    /* Retrieve the current value for the iteration counter, the value
     * of the table variable at the counter position, and the pulse size
     * from the database.
     */
    counter(Iter),
    variable(Iter,Val),
    size(Pulse),!,

    do_perturbation(Iter,Val,Pulse).

/* Add a positive pulse to the variable value and then run the model to
 * test for improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!min(Minval),
    max(Maxval),

    /* Apply the positive pulse and run the model with the new values */
    NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
    asserta(result(NewC)),!.

/* If the above rule fails then apply a negative pulse and then run the model
 * to test for improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!min(Minval),
    max(Maxval),

     /* Apply the negative pulse and run the model with the new values */
    NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
```

```
    asserta(result(NewC)),!.

/* If both of the above rules fail then return the table variable to its
 * original value.
 */
do_perturbation(Iter,Val,Pulse) :-
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)).

/* Rule initial_run initializes the variables needed for the program  and
 * runs the model for the base condition.
 */
initial_run :-
    /* Retrieve the table values into the array Vallist and store it in the
     * database as base1.
     */
    get_values(Vallist),
    asserta(base1(Vallist)),

    /* Initialize base2 and temp to the list [0] and Iter to 0.  */
    asserta(base2([0])),
    asserta(temp([0])),
    Iter is 0,

    /* Run the model for the initial values and store the result in the
     * database */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    asserta(basecummd(Cummd)),
    asserta(result(Cummd)),
    output_cummd(Cummd,Iter).

/* Rule store_temp stores the current variable values in a list called temp */

store_temp :-
    get_values(Vallist),
    abolish(temp/1),
    asserta(temp(Vallist)).

/* Rule already_at_base1 succeeds if the current table variable values are
 * equal to the values stored in the base1 list.
 */
already_at_base1 :-
    /* Retrieve the list of values assigned to base1 */
    [!base1(Base1Vallist),
    /* Retrieve the current table values into the list Vallist. */
    get_values(Vallist)!],
    /* Rule same succeeds if the two lists are the same. */
    same(Base1Vallist,Vallist).
```

```
/* Rule retrieve_base1_values resets the values of the table variable to
 * the values stored in the list assigned to base1. The model is then run
 * to get the value for CUMMD at base1.
 */
retrieve_base1_values :-
    /* Retract the old values from the database  */
    abolish(variable/2),

    /* Retrieve the list of values for base1 from the database. */
    base1(Base1Vallist),

    /* Iteratively assign each of the values in the list to the corresponding
     * table element.
     */
    asserta(count(0)),
    retrieve_base1_val_iter(Base1Vallist),
    abolish(count/1),

    /* Run the model at the base1 values and save the result.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    abolish(result/1),
    asserta(result(Cummd)),
    output_cummd(Cummd,0).

/* Rule retrieve_base1_val_iter performs the iteration to assign each element
 * of the base1 list to the table variable.
 */

/* The basis case to end the recursion when the list is empty.  */
retrieve_base1_val_iter([]).

/* Rule retrieve base1_val_iter stores the first element of the list to the
 * table variable element NewI and then recurses with the remainder of the
 * list.
 */
retrieve_base1_val_iter([X|L]) :-
    count(I),
    NewI is I + 1,
    retract(count(I)),
    asserta(count(NewI)),
    asserta(variable(NewI,X)),
    retrieve_base1_val_iter(L).


/* Rule jump_pattern assigns the value of the new temporary head the base2
 * values plus the difference between the base2 and base1 values in
 * accordance with the pattern search algorithm.
 */
jump_pattern :-
```

```
    /* Retrieve the current base1 and base2 value lists */
    base1(Base1Vallist),
    base2(Base2Vallist),

    min(Minval),
    max(Maxval),
    abolish(variable/2),
    asserta(count(0)),
    /* Calculate the new values for the temporary head */
    calc_new_vals(Base1Vallist,Base2Vallist,Minval,Maxval),
    abolish(count/1),

    /* Run the model at the temporary head position.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    abolish(result/1),
    asserta(result(Cummd)),
    output_cummd(Cummd,0).

/* Rule calc_new_vals calculates the new values for the temporary head
 * resulting from a pattern jump.
 */

/* The basis case to end the recursion when the two lists are empty. */
calc_new_vals([],[],Minval,Maxval).

/* Calculate the new value by adding to the first element of the second list
 * the difference between the first element of the second list and the first
 * element of the first list. The rule then recurses on the remainder of the
 * lists until both lists are empty.
 */
calc_new_vals([Val1|L1],[Val2|L2],Minval,Maxval) :-
    count(I),
    NewI is I + 1,
    retract(count(I)),
    asserta(count(NewI)),
    /* Calculate the new value for the ith element of the table */
    Tval is Val2 + (Val2 - Val1),

    /* Add the new value to the database and then recurse on the remainder
     * of the two lists.
     */
    range_check(Tval,Maxval,Minval,Val),
    asserta(variable(NewI,Val)),
    calc_new_vals(L1,L2,Minval,Maxval).

/* Rule get_cummd runs the model with the new value Val and then reads the
 * new value for CUMMD from the file project.out.
 */
get_cummd(Val,Iter,Cummd) :-
```

```
    /* Replace the old variable value in the database with the new value. */
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)),

    /* Run the model and read the new value for CUMMD.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    output_cummd(Cummd,Iter).

/* Rule get_values returns all the current values of the table variable
 * in the list Vallist.
 */
get_values(Vallist) :-
    tabsize(TabSize),
    asserta(count(TabSize)),
    get_val_iter([],Vallist,TabSize),
    abolish(count/1).

/* Rule get_val_iter gets each of the table variable elements from the
 * database and returns them in the list of the second argument.
 */

/* The basis case to end the recursion when I is 0.  */
get_val_iter(Vallist1,Vallist1,I) :-
    I == 0.

/* Get each element of the table variable starting with the last and append
 * it to the list Vallist1. The resulting list is returned in Vallist2.
 */
get_val_iter(Vallist1,Vallist2,I) :-
    variable(I,Val),
    append([Val],Vallist1,Vallist3),
    NewI is I - 1,
    get_val_iter(Vallist3,Vallist2,NewI).

/* Rule exit_condition checks to see if the pulse size is less than the
 * user specified minimum and if so writes the final value for CUMMD to
 * the file summary.dat and then runs the model at the final values before
 * exiting the program. The model is run so the final version of the file
 * project.rsl contains information on the final result of the optimization.
 */
exit_condition :-
    /* Determine if the pulse size if less than the minimum.  */
    [!size(Pulse),
    minsize(MinPulse)!],
    Pulse < MinPulse,

    /* If so then write the final values to the file summary.dat, run the
     * model with the final values and then quit.
     */
```

```
    open(S,'summary.dat',a),
    result(FinalCummd),
    write(S,'The final cummd is: '),write(S,FinalCummd),nl(S),
    close(S),

    output_values,
    shell(model),

    halt.

/* Rule read_cummd reads the new value of CUMMD from the file project.out
 * and stores it in the database after a simulation has been run.
 */
read_cummd :-
    open(C,'project.out',r),
    read(C,Output),
    abolish(cummd/1),
    asserta(Output),
    close(C).

/* Rule output_cummd writes the value of Cummd to the file summary.dat. */
output_cummd(Cummd,Iter) :-
    open(S,'summary.dat',a),
    write(S,Iter),write(S,'. '),
    write(S,'CUMMD= '),
    write(S,Cummd),nl(S),
    close(S).

/* Rule output_values outputs the values of the table variable to project.dnx
 * for use in the simulation and summary.dat for post-run analysis.
 */
output_values :-
    var(VarName),
    tabsize(TabSize),

    /* Write the values to project.dnx  */
    create(D,'project.dnx'),
    write(D,'T '), write(D,VarName), write(D,'='),
    foriterate(write_val1(D),TabSize),
    nl(D), close(D),

    /* Write the values to summary.dat */
    open(S,'summary.dat',a),
    write(S,VarName),write(S,'='),
    foriterate(write_val2(S),TabSize),
    nl(S), close(S).

/* Rule write_val1 is used to write the a value in the format required in
 * the file project.dnx.
 */
write_val1(F) :-
    counter(I),
```

```
    variable(I,Val),
    write(F,Val), write(F,' ').

/* Rule write_val2 is used to write a value in the format required in the
 * file summary.dat.
 */
write_val2(F) :-
    counter(I),
    variable(I,Val),
    write(F,Val), write(F,'/').

/* Rule get_initial_values prompts the user for the initial value
 * for an entry in the table and then stores that value in the
 * current element of the predicate named variable.
 */
get_initial_values :-
    counter(Iter),
    write('Enter Table Value '), write(Iter), write('. '),
    readnumber(Val),
    asserta(variable(Iter,Val)).

/* Rule output_header writes the user specified values to the file
 * summary.dat to aid in post-run analysis.
 */
output_header(VarName,TableSize,Maxval,Minval,Pulse,MinPulse) :-
    create(S,'summary.dat'),
    write(S,'       Variable Name      = '),write(S,VarName),nl(S),
    write(S,'       Table Size         = '),write(S,TableSize),nl(S),
    write(S,'       Maximum Value      = '),write(S,Maxval),nl(S),
    write(S,'       Minimum Value      = '),write(S,Minval),nl(S),
    write(S,'       Pulse size factor  = '),write(S,Pulse),nl(S),
    write(S,'       Minimum Pulse Size = '),write(S,MinPulse),nl(S),
    nl(S),
    close(S).

/* Rule output_cummd_info writes the old and new values for CUMMD
 * to the file summary.dat.
 */
output_cummd_info(OldC,NewC) :-
    open(S, 'summary.dat',a),
    nl(S),write(S, 'Base CUMMD is '),write(S,OldC),nl(S),
    write(S,'Final Temp Head CUMMD is '),write(S,NewC),nl(S),
    close(S).

/************************** Utility Routines ****************************/

/* Rule foriterate performs iteration on the predicate Pred from 1 to N
 * similar to a for loop in Pascal or C.
 */
foriterate(Pred,N) :-
    asserta(counter(0)),
    repeat,
```

```
        [!counter(K),
         K2 is K + 1,
         retract(counter(K)),
         asserta(counter(K2)),
         iterate2(Pred)!],
    K2 >= N,
    retract(counter(K2)),!.

/* Rule iterate2 is used by foriterate to execute the predicate Pred. */

iterate2(Pred) :-
    (call(Pred),!), 1=1.

/* Rule output_message writes the message passed to it to the file
 * summary.dat.
 */
output_message(Message) :-
    open(S,'summary.dat',a),
    nl(S),write(S,'**** '),write(S,Message),write(S,' ****'),nl(S),
    close(S).

/* Rule range_check ensures that the input variable is between the
 * minimum and maximum value for that variable and if not assigns
 * the minimum or maximum value to the result.
 */

/* If the value is greater than the maximum then return the maximum value */
range_check(Val,Max,Min,Result) :-
    Val > Max,
    Result is Max,!.

/* If the value is less than the minimum then return the minimum value.  */
range_check(Val,Max,Min,Result) :-
    Val < Min,
    Result is Min,!.

/* Otherwise return the initial value Val.  */
range_check(Val,Max,Min,Result) :-
    Result is Val,!.

/* Rule same succeeds if the two lists are exactly equal.  */

same([],[]).

/* Note-- the snip operators ([!,!]) are not standard prolog. */
same([X|L1],[Y|L2]) :-
    [!same(L1,L2)!],
    X == Y.

/* Rule append appends the second list to the first list and returns
 * the new list in the third argument.
 */
```

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
    append(L1,L2,L3).

/* Rule readnumber reads in a value from the user as a string and then
 * converts it to a prolog number.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and float_text are built in Arity prolog predicates and are not
 * standard prolog.
 */

readnumber(Result) :-
    read_string(255,ArString),nl,
    float_text(Result,ArString,_).

readnumber(Result) :-
    write('Number expected, try again'),
    nl,
    readnumber(Result).

/* Rule readinteger reads in a value from the user as a string and then
 * converts it to a prolog integer.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and int_text are built in Arity prolog predicates and are not
 * standard prolog.
 */

readinteger(Result) :-
    read_string(255,ArString),nl,
    int_text(Result,ArString).

readinteger(Result) :-
    write('Integer expected, try again'),
    nl,
    readnumber(Result).
```

APPENDIX C

```
/*************************************************************************
 * PROGRAM : DEPTH.ARI
 * AUTHOR  : LT C E AGAN
 * ADVISOR : PROFESSOR T. K. ABDEL-HAMID
 * DATE    : 1/10/90
 * REVISION: 2/9/90
 * SYSTEM  : IBM compatible PC using ARITY Prolog Interpreter
 * PURPOSE : This program is designed to be used with the DYNAMO Model
 * for Software Project Management to find the minimum cost, measured in
 * cumulative man-days (the variable CUMMD in the DYNAMO Model), for a
 * given project by using depth first search on two variables. The search is
 * accomplished by first adjusting the value of a single table variable then
 * a single constant variable. The name of the variables along with initial
 * values and pulse sizes are supplied by the user. A description of the
 * algorithms used for the search are given later in the code.
 *************************************************************************/

/*************************************************************************
 *                      PROGRAM REQUIREMENTS
 *
 *     This program was written for use with an Arity Prolog Interpreter.
 * Comments are included in the code where non-standard (Clocksin-Mellish)
 * predicates were used. These code segments may require changes before
 * running under a different prolog system.
 *     The program requires that the following files be in the same
 * directory as the program when it is executed:
 *     1. PROJECT.DYN - The dynamo file containing the program for the
 *        project to be simulated (and all the other files with the names
 *        project.* that are required by dynamo to run a simulation).
 *     2. PROJECT.DRS - The Dynamo report format file that is used to
 *        write the result of the simulation to the file PROJECT.OUT for
 *        the program to read (PROJECT.OUT will be created and need not
 *        exist prior to running the program).
 *     3. MODEL.BAT - A batch file containing the DOS commands to simulate
 *        the project using DYNEX and to output the results (This file calls
 *        the following EXE files that must also be in the default directory:
 *        DYNEX.EXE, SMLT.EXE, REP.EXE. These three files are all part of
 *        the Professional Dynamo Simulator Version 3.3).
 *************************************************************************/

/*************************************************************************
 *                      INSTRUCTIONS
 *
 *     To run this program first initiate the Arity Prolog Interpreter by
 *     typing API at the DOS prompt. When the prolog prompt (?-) appears
```

```
 *     type [depth]. (including the period) and press enter. The
 *     interpreter should then answer with "yes" and return to ?- . At this
 *     point type depth_search. and press enter. The program will then
 *     ask a series of questions to be answered by the user (such as the
 *     name of the variables to be adjusted). Enter your reply and press
 *     enter (NOTE -- you need not follow the answers with a period).
 ************************************************************************/

/*************************** SOURCE CODE ***************************/

/* Rule depth_search initiates a depth first search to optimize a
 * table function variable and then a single valued variable. The
 * table variable is first optimized using optimize_table and then
 * the single variable is optimized using optimize_single. NOTE -- to
 * reverse the order of optimization simply switch the order of execution
 * of the two rules optimize_table and optimize_single.
 */

depth_search :-
    init_vars,
    optimize_table,
    optimize_single,
    halt.

/* Rule optimize_table is the main module for the table function pattern
 * search optimization module. It first runs the simulation for the initial
 * points and then calls the rule pattern_search to execute the pattern search
 * algorithm.
 */

optimize_table :-
    initial_run,
    pattern_search.

/* Rule pattern_search executes the pattern search algorithm (Hooke and
 * Jeeves - 1961). The basic theory of the algorithm is as follows:
 *     1. Begin at an arbitrary point base1 (user supplied initial points).
 *     2. Perturb each of the variables in the table by first adding and then
 *        subtracting the pulse size from the value. If either pulse results
 *        in a value for CUMMD less than the base then it becomes the new
 *        temporary head. Otherwise the variable is returned to its original
 *        value.
 *     3. When all the perturbations are complete if value for CUMMD at the
 *        final temporary head is less than at base1 then it becomes the new
 *        base2. Since the previous move resulted in improvement, the
 *        algorithm assumes that another move in the same direction will also
 *        yield improvement so the pattern is jumped to a new temporary head.
 *        The jump is accomplished by adding to each of the values in base2
 *        the difference between the base2 and base1 values for that point.
 *        Finally base2 becomes the new base1 and return to step 2.
 *     4. If the final temporary head is not an improvement over base1 then
 *        the pattern retreats to base1 to do local perturbation around it
```

```
 *       to try to establish a new pattern.  If a new better temporary head
 *       is found then the pattern is jumped and return to step 2.
 *    5. If local explorations around the base do not result in a new pattern
 *       then the previous pattern is destroyed, the pulse size is halved,
 *       and the base1 values become the starting point for a new pattern.
 *    6. The search terminates when the pulse size falls below the user set
 *       minimum pulse size.
 */


/* Test if the exit condition is true  */
pattern_search :-
    exit_condition.

/* Case where final temporary head is better than previous base so temporary
 * head becomes the new base2 and the pattern is jumped.
 */

pattern_search :-
    /* NOTE-- the snip operators ([!,!]) are not standard prolog.  Their
     * purpose is similar to the cut operator (!) except that they only
     * prevent backtracking into the statements inside the snips and they do
     * not have the side effect of preventing execution of other predicates
     * of the same name as the one where they were executed.
     */

    /* Retrieve the value for CUMMD at the current base and the size of the
     * table from the database.
     */
    [!basecummd(OldCummd),
    tabsize(TableSize),

    /* Perturb each of the variables in the table  */
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store the values for each variable to the list temp */
    t_result(NewCummd)!], /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd),

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only be executed if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve the current values for the temp head */

    /* Set base2 equal to the temporary head and set the value basecummd
     * to NewCummd.
     */
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
```

```
    asserta(basecummd(NewCummd)),

    /* Jump the pattern to the new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.


/* Case where temp head is not better than base so try local exploration
 * about the base.
 */
pattern_search :-

    /* First test to see if already at base1.  If not retreat to base1 and
     * perform local explorations.
     */
    [!not(already_at_base1),
    output_message('Retreat to Base for Local Exploration'),
    retrieve_base1_values,

    /* Retrieve the table size and value for CUMMD at the base from the
     * database.
     */
    tabsize(TableSize),
    basecummd(OldCummd),

    /* Perturb each of the variables in the table  */
    foriterate(perturb_variables,TableSize),

    store_temp,   /* Store the values for each variable to the list temp */
    t_result(NewCummd), /* Retrieve the new value for CUMMD */
    output_cummd_info(OldCummd,NewCummd)!],

    NewCummd < OldCummd,  /* Compare the new CUMMD to the old   */

    /* The following steps will only be executed if above test succeeds */

    output_message('Jump Pattern'), /* Annotate the file summary.dat */
    temp(TempVallist), /* Retrieve the current values for the temp head */

    /* Set base2 equal to the temporary head and set the value basecummd
     * to NewCummd.
     */
    abolish(base2/1),
    asserta(base2(TempVallist)),
    retract(basecummd(OldCummd)),
    asserta(basecummd(NewCummd)),
```

```
    /* Jump the pattern to the new temporary head for local explorations */
    jump_pattern,

    /* Set base1 equal to base2 and restart the procedure.  */
    base1(Base1Vallist),
    base2(Base2Vallist),
    [!abolish(base1/1),
    asserta(base1(Base2Vallist))!],
    pattern_search.

/* Case where local explorations about base1 do not result in improvement so
 * reduce pulse size and start pattern over from base1.
 */

pattern_search :-
    output_message('Reduce Table Pulse Size'), /* Annotate summary.dat  */
    retrieve_base1_values,

    /* Reduce pulse size and start pattern search over  */
    t_size(Pulse),
    NewPulse is Pulse * 0.5,
    retract(t_size(Pulse)),
    asserta(t_size(NewPulse)),
    pattern_search.

/* Rule perturb_variable is passed to foriterate and is used to retrieve the
 * necessary variables and then call do_perturbation in order to perturb
 * each value in the table.
 */
perturb_variables :-
    /* Retrieve the current value for the iteration counter, the value
     * of the table variable at the counter position, and the pulse size
     * from the database.
     */
    counter(Iter),
    variable(Iter,Val),
    t_size(Pulse),!,

    do_perturbation(Iter,Val,Pulse).

/* Add a positive pulse to the variable value and then run the model to
 * test for improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!t_min(Minval),
    t_max(Maxval)!],
    Val \== Maxval,

    /* Apply the positive pulse and run the model with the new values */
    [!NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
```

```
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    t_result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(t_result/1),
    asserta(t_result(NewC)),!.

/* If the above rule fails then apply a negative pulse and then run the model
 * to test for improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!t_min(Minval),
    t_max(Maxval)!],
    Val \== Minval,

     /* Apply the negative pulse and run the model with the new values */
    [!NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    t_result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(t_result/1),
    asserta(t_result(NewC)),!.

/* If both of the above rules fail then return the table variable to its
 * original value.
 */
do_perturbation(Iter,Val,Pulse) :-
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)).

/* Rule store_temp stores the current variable values in a list called temp */

store_temp :-
    get_values(Vallist),
    abolish(temp/1),
    asserta(temp(Vallist)).

/* Rule already_at_base1 succeeds if the current table variable values are
 * equal to the values stored in the base1 list.
 */
```

```
already_at_base1 :-
    /* Retrieve the list of values assigned to base1 */
    [!base1(Base1Vallist),
    /* Retrieve the current table values into the list Vallist. */
    get_values(Vallist)!],
    /* Rule same succeeds if the two lists are the same. */
    same(Base1Vallist,Vallist).

/* Rule retrieve_base1_values resets the values of the table variable to
 * the values stored in the list assigned to base1. The model is then run
 * to get the value for CUMMD at base1.
 */
retrieve_base1_values :-
    /* Retract the old values from the database  */
    abolish(variable/2),

    /* Retrieve the list of values for base1 from the database. */
    base1(Base1Vallist),

    /* Iteratively assign each of the values in the list to the corresponding
     * table element.
     */
    asserta(count(0)),
    retrieve_base1_val_iter(Base1Vallist),
    abolish(count/1),

    /* Run the model at the base1 values and save the result.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    abolish(t_result/1),
    asserta(t_result(Cummd)),
    output_cummd(Cummd,0).

/* Rule retrieve_base1_val_iter performs the iteration to assign each element
 * of the base1 list to the table variable.
 */

/* The basis case to end the recursion when the list is empty.  */
retrieve_base1_val_iter([]).

/* Rule retrieve base1_val_iter stores the first element of the list to the
 * table variable element NewI and then recurses with the remainder of the
 * list.
 */
retrieve_base1_val_iter([X|L]) :-
    count(I),
    NewI is I + 1,
    retract(count(I)),
    asserta(count(NewI)),
    asserta(variable(NewI,X)),
```

```
    retrieve_base1_val_iter(L).

/* Rule jump_pattern assigns the value of the new temporary head the base2
 * values plus the difference between the base2 and base1 values in
 * accordance with the pattern search algorithm.
 */
jump_pattern :-
    /* Retrieve the current base1 and base2 value lists */
    base1(Base1Vallist),
    base2(Base2Vallist),

    t_min(Minval),
    t_max(Maxval),
    abolish(variable/2),
    asserta(count(0)),
    /* Calculate the new values for the temporary head */
    calc_new_vals(Base1Vallist,Base2Vallist,Minval,Maxval),
    abolish(count/1),

    /* Run the model at the temporary head position.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    abolish(t_result/1),
    asserta(t_result(Cummd)),
    output_cummd(Cummd,0).

/* Rule calc_new_vals calculates the new values for the temporary head
 * resulting from a pattern jump.
 */

/* The basis case to end the recursion when the two lists are empty. */
calc_new_vals([],[],Minval,Maxval).

/* Calculate the new value by adding to the first element of the second list
 * the difference between the first element of the second list and the first
 * element of the first list. The rule then recurses on the remainder of the
 * lists until both lists are empty.
 */
calc_new_vals([Val1|L1],[Val2|L2],Minval,Maxval) :-
    count(I),
    NewI is I + 1,
    retract(count(I)),
    asserta(count(NewI)),
    /* Calculate the new value for the ith element of the table */
    Tval is Val2 + (Val2 - Val1),

    /* Add the new value to the database and then recurse on the remainder
     * of the two lists.
     */
    range_check(Tval,Maxval,Minval,Val),
```

```
    asserta(variable(NewI,Val)),
    calc_new_vals(L1,L2,Minval,Maxval).

/* Rule optimize_single is the main module for the single valued variable
 * optimization module.  It first calls s_get_cummd to get the value for CUMMD
 * at the starting point and then calls minimize to find the minimum value.
 */
optimize_single :-
    s_size(Pulse),
    test(Val),
    s_get_cummd(Val,0,Cummd),
    minimize(Val,Pulse,0).

/* Rule minimize finds the minimum value of the function by using the
 * following algorithm:
 *    1. See if the exit condition has been met.  For this program the exit
 *       condition is when the pulse size is reduced below the user
 *       supplied minimum pulse size.  When the exit condition is met the
 *       rule is finished.
 *    2. Find the direction of improvement by first applying a negative pulse
 *       and comparing the resulting value of CUMMD with the original value.
 *       If the new value is greater then apply a positive pulse and perform
 *       the same test.  If neither pulse is better then the pulse size is
 *       halved and return to step 1.
 *    3. If a direction of improvement is found then continue moving in that
 *       direction until improvement is no longer noted.  The last value
 *       where improvement was found becomes the new minimum.
 *    4. Halve the pulse size and return to step 1 with the minimum point
 *       as the new starting point.
 */

/* If the exit condition is met end the rule  */

minimize(Value,Pulse,I) :-
    s_exit_condition, !.

/*  Otherwise continue to execute the search algorithm  */

minimize(Value,Pulse,I) :-
     s_min(Minval),
     s_max(Maxval),
     NewI is I + 1,

     /* Pulse in the negative direction  */
     NewValue is Value - Pulse,

     /* Ensure that the new value is not less then the minimum */
     range_check(NewValue,Maxval,Minval,ScaledVal),

     /* Retrieve the old result and run the model to find the new result */
     s_result(I,Value,OldC),
     s_get_cummd(ScaledVal,NewI,NewC),
```

```
    /* If the negative pulse was an improvement then set the value of Pulse
     * to Pulse * -1, otherwise pulse in the positive direction
     * NOTE-- the ifthenelse predicate is not standard prolog
     */
    ifthenelse(NewC =< OldC,pulse_neg(Pulse),
                            pulse_pos(Value,Pulse,NewI)),

    /* Continue in the direction of improvement starting at the last
     * point tested by calling the rule find_min
     */
    s_size(NewPulse),
    s_result(NewI1,NewValue1,NewC1),
    find_min(NewValue1,NewPulse,NewI1,NewC1,OldC),

    /* Retract the result from the point where improvement was no longer
     * noted */
    retract(s_result(A,B,C)),

    /* Retrieve the result from the last point of improvement and assert it
     * as the new minimum
     */
    s_result(MinI,MinValue,MinCummd),
    asserta(minimum(MinValue,MinCummd)),
    output_message('Reduce single variable pulse size '),

    /* Halve the Pulse size and return to the beginning of the rule */
    SmallerPulse is Pulse * 0.5,
    abolish(s_size/1),
    asserta(s_size(SmallerPulse)),
    minimize(MinValue,SmallerPulse,MinI).

/* Rule pulse_neg is called when an initial negative pulse yields improvement
 * over the previous value. Since the value of pulse is always added to the
 * previous value of the variable, this rule changes the sign of Pulse to
 * negative in order to have the effect of subtraction.
 */
pulse_neg(Pulse) :-
    NewPulse is Pulse * -1,
    abolish(s_size/1),
    asserta(s_size(NewPulse)).

/* Rule pulse_pos is called whenever a negative pulse did not yield
 * improvement. First the result from the negative pulse is retracted,
 * then a positive pulse is added to the original value. Rule get_cummd
 * is then called to find the value of CUMMD at the new point.
 */
pulse_pos(Value,Pulse,I) :-
    s_min(Minval),
    s_max(Maxval),
    retract(s_result(I,V,X)),
    NewValue is Value + Pulse,
```

```
    range_check(NewValue,Maxval,Minval,ScaledVal),
    s_get_cummd(ScaledVal,I,Cummd).

/* Rule find_min performs the iteration that continues to pulse in the
 * given direction until it no longer leads to improvement in CUMMD.
 */

/* If the new CUMMD is more than the old CUMMD then exit the rule */
find_min(Value,Pulse,I,NewC,OldC) :-
    NewC > OldC,!.

/* If the current pulse is positive and the variable is at the maximum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    s_max(Maxval),
    Pulse > 0,
    Value == Maxval,
    asserta(s_result(I,Value,NewC)),!.

/* If the current pulse is negative and the variable is at the minimum
 * value then exit the rule.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    s_min(Minval),
    Pulse < 0,
    Value == Minval,
    asserta(s_result(I,Value,NewC))!.

/* Otherwise pulse in the current direction of the variable Pulse and
 * then run the model to get the new value of CUMMD.
 */
find_min(Value,Pulse,I,NewC,OldC) :-
    s_min(Minval),
    s_max(Maxval),

    /* Pulse in the previously set direction of improvement */
    NewVal is Value + Pulse,

    /* Ensure that the new value is between the user supplied minimum
     * and maximum value
     */
    range_check(NewVal,Maxval,Minval,ScaledVal),

    /* Get the new value of CUMMD and iterate the rule through the
     * use of tail recursion.
     */
    NewI is I + 1,
    s_get_cummd(ScaledVal,NewI,Cummd),
    find_min(ScaledVal,Pulse,NewI,Cummd,NewC).
```

```
/* Rule init_vars prompts the user for the initial values used in
 * the program and stores the values in the database.
 */
init_vars :-
    write('What table variable do you want to optimize? '),
    read_string(30,String_TVarName),nl,
    atom_string(TVarName,String_TVarName),
    asserta(t_var(TVarName)),
    write('How many values are in your table? '),
    readinteger(TableSize),
    asserta(tabsize(TableSize)),
    write('What is the maximum value for a variable in the table? '),
    readnumber(TMaxval),
    asserta(t_max(TMaxval)),
    write('What is the minimum value for a variable in the table? '),
    readnumber(TMinval),
    asserta(t_min(TMinval)),
    write('What is your desired pulse size for this variable? '),
    readnumber(TPulse),
    asserta(t_size(TPulse)),
    write('What is the minimum pulse size for this variable? '),
    readnumber(TMinPulse),
    asserta(t_minsize(TMinPulse)),
    foriterate(get_initial_values,TableSize),

    write('What single value variable do you want to optimize? '),
    read_string(30,ArString),nl,
    atom_string(SVarName,ArString),
    asserta(s_var(SVarName)),
    write('What is the maximum value for the variable? '),
    readnumber(SMaxval),
    asserta(s_max(SMaxval)),
    write('What is the minimum value for the variable? '),
    readnumber(SMinval),
    asserta(s_min(SMinval)),
    write('What is your desired pulse size for this variable? '),
    readnumber(SPulse),
    asserta(s_size(SPulse)),
    write('What is your minimum pulse size for this variable? '),
    readnumber(SMinPulse),
    asserta(s_minsize(SMinPulse)),
    write('Enter the starting value for the variable. '),
    readnumber(Val),
    asserta(test(Val)),
    asserta(s_val(Val)),
    output_header(TVarName,SVarName,TPulse,TMinPulse,SPulse,SMinPulse).

/* Rule initial_run initializes the variables needed for the program
 * and runs the model for the base condition.
 */
initial_run :-
    /* Retrieve the table values into the array Vallist and store it in the
```

```
     * database as base1.
     */
    get_values(Vallist),
    asserta(base1(Vallist)),

    /* Initialize base2 and temp to the list [0] and Iter to 0.  */
    asserta(base2([0])),
    asserta(temp([0])),
    Iter is 0,

    /* Run the model for the initial values and store the result in the
     * database */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    asserta(basecummd(Cummd)),
    asserta(t_result(Cummd)),
    output_cummd(Cummd,Iter).

/* Rule get_cummd runs the model with the new value Val and then reads the
 * new value for CUMMD from the file project.out.
 */
get_cummd(Val,Iter,Cummd) :-
    /* Replace the old variable value in the database with the new value. */
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)),

    /* Run the model and read the new value for CUMMD.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    output_cummd(Cummd,Iter).

/* Rule s_get_cummd first tests to see if a result already exists for the
 * given value.  If not the model is run to find the value for CUMMD at
 * the given value for the test variable.
 */

/* If the result already exists in the database then set it to the current
 * result.
 */
s_get_cummd(Value,I,Cummd) :-
    s_result(X,Value,Cummd),
    abolish(s_val/1),
    asserta(s_val(Value)),
    output_values,
    output_cummd(Cummd,I),
    asserta(s_result(I,Value,Cummd)).

/* Otherwise run the model to find the value for CUMMD at the given Value */
```

```
s_get_cummd(Value,I,Cummd) :-
    abolish(s_val/1),
    asserta(s_val(Value)),
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    output_cummd(Cummd,I),
    asserta(s_result(I,Value,Cummd)).

/* Rule get_values returns all the current values of the variable
 * in the list Vallist.
 */
get_values(Vallist) :-
    tabsize(TabSize),
    asserta(count(TabSize)),
    get_val_iter([],Vallist,TabSize),
    abolish(count/1).

/* Rule get_val_iter gets each of the table variable elements from the
 * database and returns them in the list of the second argument.
 */

/* The basis case to end the recursion when I is 0.  */
get_val_iter(Vallist1,Vallist1,I) :-
    I == 0.

/* Get each element of the table variable starting with the last and append
 * it to the list Vallist1. The resulting list is returned in Vallist2.
 */
get_val_iter(Vallist1,Vallist2,I) :-
    variable(I,Val),
    append([Val],Vallist1,Vallist3),
    NewI is I - 1,
    get_val_iter(Vallist3,Vallist2,NewI).

/* Rule exit_condition checks to see if the pulse size is less than the
 * user specified minimum and if so writes the final value for CUMMD to
 * the file summary.dat.
 */
exit_condition :-
    /* Determine if the pulse size if less than the minimum.  */
    [!t_size(Pulse),
    t_minsize(MinPulse)!],
    Pulse < MinPulse,

    /* If so then write the final values to the file summary.dat.  */
    open(S,'summary.dat',a),
    t_result(MinCummd),
    nl(S),write(S,'The min cummd after optimizing table function is: '),
    write(S,MinCummd),nl(S),
    close(S).
```

```
/* Rule s_exit_condition succeeds if the size of Pulse is less than the
 * minimum pulse size given by the user.  If the rule succeeds the model is
 * run with the final values so that the results in the file project.rsl will
 * reflect the optimal condition.
 */
s_exit_condition :-
    /* Determine if the pulse size is less than the minimum  */
    [!s_size(Pulse),
    s_minsize(MinPulse)!],
    Pulse < MinPulse,

    /* If the pulse size is less than the minimum then execute the model
     * using the points that resulted in the minimum value for CUMMD.
     */
    minimum(Minval,MinCummd),
    abolish(s_val/1),
    asserta(s_val(Minval)),
    output_message('The final values are '),
    output_values,
    shell(model),
    output_cummd(MinCummd,0).

/* Rule read_cummd reads the new value of CUMMD from the file project.out
 * and stores it in the database after a simulation has been run.
 */
read_cummd :-
    open(C,'project.out',r),
    read(C,Output),
    abolish(cummd/1),
    asserta(Output),
    close(C).

/* Rule output_cummd writes the value of Cummd to summary.dat */
output_cummd(Cummd,Iter) :-
    open(S,'summary.dat',a),
    write(S,Iter),write(S,'. '),
    write(S,'CUMMD= '),
    write(S,Cummd),nl(S),
    close(S).

/* Rule output_values outputs the values of the variable to project.dnx for
 * use in the simulation and summary.dat for post-run analysis.
 */
 output_values :-
    t_var(TVarName),
    tabsize(TabSize),
    s_val(SVal),
    s_var(SVarName),

    /* Write the values to project.dnx  */
    create(D,'project.dnx'),
```

```
    write(D,'T '), write(D,TVarName), write(D,'='),
    foriterate(write_val1(D),TabSize),nl(D),
    write(D,'C '), write(D,SVarName), write(D,'='),
    write(D,SVal),nl(D),
    close(D),

    /* Write the values to summary.dat */
    open(S,'summary.dat',a),
    write(S,TVarName),write(S,'='),
    foriterate(write_val2(S),TabSize),nl(S),
    write(S,SVarName), write(S,'='), write(S,SVal),
    nl(S), close(S).

/* Rule write_val1 is used to write the a value in the format required in
 * the file project.dnx.
 */
write_val1(F) :-
    counter(I),
    variable(I,Val),
    write(F,Val), write(F,' ').

/* Rule write_val2 is used to write a value in the format required in the
 * file summary.dat.
 */
write_val2(F) :-
    counter(I),
    variable(I,Val),
    write(F,Val), write(F,'/').

/* Rule get_initial_values prompts the user for the initial value
 * for an entry in the table and then stores that value in the
 * current element of the predicate named variable.
 */
get_initial_values :-
    counter(Iter),
    write('Enter Table Value '), write(Iter), write('. '),
    readnumber(Val),
    asserta(variable(Iter,Val)).


/* Rule output_header writes the user specified values to the file
 * summary.dat to aid in post-run analysis.
 */
output_header(TVarName,SVarName,TPulse,TMinPulse,SPulse,SMinPulse) :-
    create(S,'summary.dat'),
    write(S,'       Table Variable Name       = '),write(S,TVarName),nl(S),
    write(S,'       Single Variable Name      = '),write(S,SVarName),nl(S),
    write(S,'       Table Pulse size factor   = '),write(S,TPulse),nl(S),
    write(S,'       Table Minimum Pulse Size  = '),write(S,TMinPulse),nl(S),
    write(S,'       Single Pulse size factor  = '),write(S,SPulse),nl(S),
    write(S,'       Single Minimum Pulse Size = '),write(S,SMinPulse),nl(S),
    nl(S),
```

```
    close(S).

/*  Rule output_cummd_info writes the old and new values for CUMMD
 *  to summary.dat.
 */
output_cummd_info(OldC,NewC) :-
    open(S, 'summary.dat',a),
    nl(S),write(S, 'Base CUMMD is '),write(S,OldC),nl(S),
    write(S,'Final Temp Head CUMMD is '),write(S,NewC),nl(S),
    close(S).

/************************ Utility Routines ***********************************/

/* Rule foriterate performs iteration on the predicate Pred from 1 to N
 * similar to a for loop in Pascal or C.
 */
foriterate(Pred,N) :-
    asserta(counter(0)),
    repeat,
          [!counter(K),
           K2 is K + 1,
           retract(counter(K)),
           asserta(counter(K2)),
           iterate2(Pred)!],
    K2 >= N,
    retract(counter(K2)),!.

/* Rule iterate2 is used by foriterate to execute the predicate Pred. */

iterate2(Pred) :-
    (call(Pred),!), 1=1.

/*  Rule output_message writes the message passed to it to the
 *  file summary.dat.
 */
 output_message(Message) :-
    open(S,'summary.dat',a),
    nl(S),write(S,'**** '),write(S,Message),write(S,' ****'),nl(S),
    close(S).

/* Rule range_check ensures that the input variable is between the
 * minimum and maximum value for that variable and if not assigns
 * the minimum or maximum value to the result.
 */

/* If the value is greater than the maximum then return the maximum value */
range_check(Val,Max,Min,Result) :-
    Val > Max,
    Result is Max,!.

/* If the value is less than the minimum then return the minimum value.  */
range_check(Val,Max,Min,Result) :-
```

```
    Val < Min,
    Result is Min,!.

/* Otherwise return the initial value Val.  */
range_check(Val,Max,Min,Result) :-
    Result is Val,!.

/* Rule same succeeds if the two lists are exactly equal.  */

same([],[]).

/* Note-- the snip operators ([!,!]) are not standard prolog. */
same([X|L1],[Y|L2]) :-
    [!same(L1,L2)!],
    X == Y.

/* Rule append appends the second list to the first list and returns
 * the new list in the third argument.
 */

append([],L,L).

append([X|L1],L2,[X|L3]) :-
    append(L1,L2,L3).

/* Rule readnumber reads in a value from the user as a string and then
 * converts it to a prolog number.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and float_text are built in Arity prolog predicates and are not
 * standard prolog.
 */

readnumber(Result) :-
    read_string(255,ArString),nl,
    float_text(Result,ArString,_).

readnumber(Result) :-
    write('Number expected, try again'),
    nl,
    readnumber(Result).

/* Rule readinteger reads in a value from the user as a string and then
 * converts it to a prolog integer.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and int_text are built in Arity prolog predicates and are not
 * standard prolog.
 */

readinteger(Result) :-
    read_string(255,ArString),nl,
    int_text(Result,ArString).
```

```
readinteger(Result) :-
    write('Integer expected, try again'),
    nl,
    readnumber(Result).
```

APPENDIX D

```
/*************************************************************************
 * PROGRAM : BREADTH.ARI
 * AUTHOR  : LT C E AGAN
 * ADVISOR : PROFESSOR T. K. ABDEL-HAMID
 * DATE    : 1/15/90
 * REVISION: 2/20/90
 * SYSTEM  : IBM compatible PC using ARITY Prolog Interpreter
 * PURPOSE : This program is designed to be used with the DYNAMO Model
 * for Software Project Management to find the minimum cost, measured in
 * cumulative man-days (the variable CUMMD in the DYNAMO Model), for a
 * given project by using breadth-first search on two variables. The search is
 * accomplished by first adjusting the value of a single table variable for one
 * cycle then a single constant variable also for one cycle. The name of the
 * variables along with initial values and pulse sizes are supplied by the user.
 * A description of the algorithms used for the search are given later in the code.
 *************************************************************************/


/*************************************************************************
 *                    PROGRAM REQUIREMENTS
 *
 *     This program was written for use with an Arity Prolog Interpreter.
 * Comments are included in the code where non-standard (Clocksin-Mellish)
 * predicates were used.  These code segments may require changes before
 * running under a different prolog system.
 *     The program requires that the following files be in the same
 * directory as the program when it is executed:
 *     1. PROJECT.DYN - The dynamo file containing the program for the
 *        project to be simulated (and all the other files with the names
 *        project.* that are required by dynamo to run a simulation).
 *     2. PROJECT.DRS - The Dynamo report format file that is used to
 *        write the result of the simulation to the file PROJECT.OUT for
 *        the program to read (PROJECT.OUT will be created and need not
 *        exist prior to running the program).
 *     3. MODEL.BAT - A batch file containing the DOS commands to simulate
 *        the project using DYNEX and to output the results (This file calls
 *        the following EXE files that must also be in the default directory:
 *        DYNEX.EXE, SMLT.EXE, REP.EXE.  These three files are all part of
 *        the Professional Dynamo Simulator Version 3.3).
 *************************************************************************/

/*************************************************************************
 *                    INSTRUCTIONS
 *
 *     To run this program first initiate the Arity Prolog Interpreter by
 *     typing API at the DOS prompt.  When the prolog prompt (?-) appears
 *     type [breadth]. (including the period) and press enter.  The
```

```
 *      interpreter should then answer with "yes" and return to ?- . At this
 *      point type breadth_search. and press enter. The program will then
 *      ask a series of questions to be answered by the user (such as the
 *      name of the variables to be adjusted). Enter your reply and press
 *      enter (NOTE -- you need not follow the answers with a period).
 ************************************************************************/

/*************************** SOURCE CODE ****************************/

/*  Rule breadth_search first calls inti_vars to get the intial value for the
 *  variables from the user and then calls intial run to find the value for CUMMD
 *  at the starting point. Finally the rule do_breadth_search is called to
 *  execute the breadth-first search.
 */

breadth_search :-
    init_vars,
    initial_run,
    do_breadth_search.

/*  Rule do_breadth_search executes the breadth first search by
 *  first perturbing the table values for one cycle and then perturbing
 *  the single variable value for a cycle and comparing the result with
 *  the value for CUMMD from the previous cycle to see if improvement
 *  was found. If improvement is not found then the pulse size for both
 *  variables is halved and the algorithm starts over again. The program
 *  terminates when the pulse size for both variables is reduced below the
 *  user entered minimum value.
 */

/* If the exit condition has been met then exit the rule */
do_breadth_search :-
    exit_condition.

/* Otherwise carry out the breadth-first search by first applying the
 * operator perturb_table_variable to run one cycle of adjustments on the
 * table variable and then applying the rule perturb_single_variable to
 * test the constant. If a better value for CUMMD is found then
 * the rule calls itself to continue the algorithm.
 */
do_breadth_search :-
    /* retrieve the table size and previous value for CUMMD from the
     * database.
     */
    [!tabsize(TableSize),
    result(OldCummd),

    /* Perturb each of the values in the table function */
    foriterate(perturb_table_variable,TableSize),

    /* Perturb the value of the single valued constant */
    test(Val),
```

```
    perturb_single_variable(Val),

    /* Retrieve the new value for CUMMD from the database and compare
     * it with the previous value.
     */
    result(NewCummd)!],
    NewCummd < OldCummd,

    /* If the above test succeeds then store the current values in the
     * database as the new minimum point and start another cycle.
     */
    store_min_values,
    do_breadth_search.

/* If adjustments to both variables fail to find a better value for CUMMD
 * then reduce both pulse sizes and restart the search.
 */
do_breadth_search :-
    /* Retrieve the current pulse sizes from the database */
    t_size(TPulse),
    s_size(SPulse),

    /* Halve both pulse sizes and store the new values in the database */
    NewTPulse is TPulse * 0.5,
    NewSPulse is SPulse * 0.5,
    abolish(t_size/1),
    abolish(s_size/1),
    asserta(t_size(NewTPulse)),
    asserta(s_size(NewSPulse)),

    /* Restart the search with the smaller pulse sizes */
    do_breadth_search.


/* Rule perturb_table_variable is passed to foriterate and is used to retrieve
 * the necessary variables and then call do_perturbation in order to perturb
 * each value in the table.
 */
perturb_table_vari·'  :-
    /* Retrieve the   ent value for the iteration counter, the value
     * of the table variable at the counter position, and the pulse size
     * from the database.
     */
    counter(Iter),
    variable(Iter,Val),
    t_size(Pulse),!,

    do_perturbation(Iter,Val,Pulse).

/* Add a positive pulse to the variable value and then run the model to
 * test for improvement.
 */
```

```
do_perturbation(Iter,Val,Pulse) :-
    [!t_min(Minval),
    t_max(Maxval)!],
    Val \== Maxval,

    /* Apply the positive pulse and run the model with the new values */
    [!NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
    asserta(result(NewC)),!.

/* If the above rule fails then apply a negative pulse and then run the model
 * to test for improvement.
 */
do_perturbation(Iter,Val,Pulse) :-
    [!t_min(Minval),
    t_max(Maxval)!],
    Val \== Minval,

     /* Apply the negative pulse and run the model with the new values */
    [!NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    get_cummd(ScaledVal,Iter,NewC),

    /* Retrieve the result from the pervious run from the database */
    result(OldC)!],

    /* Test if the new CUMMD is better than the previous value */
    NewC < OldC,

    /* Store the new value for CUMMD in the database. */
    abolish(result/1),
    asserta(result(NewC)),!.

/* If both of the above rules fail then return the table variable to its
 * original value.
 */
do_perturbation(Iter,Val,Pulse) :-
    retract(variable(Iter,OldVal)),
    asserta(variable(Iter,Val)).

/* Rule perturb_single variable is similar to perturb_table_variable.  First
 * a positive pulse is added to the current variable value and the model is run
```

```
 * to test for improvement in the value of CUMMD.  If improvement is not found
 * then a negative pulse is subtracted.  If neither case results in improvement
 * then the variable is returned to its original value.
 */

/* Add a positive pulse to the variable and then run the model to
 * test for improvement.  If the value is already at the maximum
 * before the positive pulse then the rule fails and the next case
 * is executed.
 */
perturb_single_variable(Val) :-
    [!s_min(Minval),
    s_max(Maxval),
    s_size(Pulse)!],

    /* If the variable is already at the maximum value then fail */
    Val \== Maxval,

    /* Add a positive pulse and run the model */
    [!NewVal is Val + Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    s_get_cummd(ScaledVal,1,NewC),
    result(OldC)!],

    /* If the new CUMMD is less than the old then store the result
     * in the database.
     */
    NewC < OldC,
    abolish(result/1),
    asserta(result(NewC)).

/* If the above rule fails then apply a negative pulse and run
 * the model again.  If the variable is already at the minimum value
 * before the negative pulse then the rule fails and the next case is
 * executed.
 */
perturb_single_variable(Val) :-
    [!s_min(Minval),
    s_max(Maxval),
    s_size(Pulse)!],

    /* If the variable is already at the minimum value then fail */
    Val \== Minval,

    /* Apply the negative pulse and run the model. */
    [!NewVal is Val - Pulse,
    range_check(NewVal,Maxval,Minval,ScaledVal),
    s_get_cummd(ScaledVal,2,NewC),
    result(OldC)!],

    /* If the new CUMMD is better than the old CUMMD then store the
     * result in the database.
```

```
    */
    NewC < OldC,
    abolish(result/1),
    asserta(result(NewC)).

/* If both the above cases fail then return the variable to its
 * original value.
 */
perturb_single_variable(Val) :-
    abolish(test/1),
    asserta(test(Val)).


/* Rule init_vars prompts the user for the initial values used in
 * the program and stores the values in the database.
 */

init_vars :-
    write('What table variable do you want to optimize? '),
    read_string(30,String_TVarName),nl,
    atom_string(TVarName,String_TVarName),
    asserta(t_var(TVarName)),
    write('How many values are in your table? '),
    readinteger(TableSize),
    asserta(tabsize(TableSize)),
    write('What is the maximum value for a variable in the table? '),
    readnumber(TMaxval),
    asserta(t_max(TMaxval)),
    write('What is the minimum value for a variable in the table? '),
    readnumber(TMinval),
    asserta(t_min(TMinval)),
    write('What is your desired pulse size for this variable? '),
    readnumber(TPulse),
    asserta(t_size(TPulse)),
    write('What is the minimum pulse size for this variable? '),
    readnumber(TMinPulse),
    asserta(t_minsize(TMinPulse)),
    foriterate(get_initial_values,TableSize),

    write('What single value variable do you want to optimize? '),
    read_string(30,ArString),nl,
    atom_string(SVarName,ArString),
    asserta(s_var(SVarName)),
    write('What is the maximum value for the variable? '),
    readnumber(SMaxval),
    asserta(s_max(SMaxval)),
    write('What is the minimum value for the variable? '),
    readnumber(SMinval),
    asserta(s_min(SMinval)),
    write('What is your desired pulse size for this variable? '),
    readnumber(SPulse),
    asserta(s_size(SPulse)),
```

```
    write('What is your minimum pulse size for this variable? '),
    readnumber(SMinPulse),
    asserta(s_minsize(SMinPulse)),
    write('Enter the starting value for the variable. '),
    readnumber(Val),
    asserta(test(Val)),
    asserta(s_val(Val)),
    output_header(TVarName,SVarName,TPulse,TMinPulse,SPulse,SMinPulse).

/* Rule initial_run initializes the variables needed for the program
 * and runs the model for the base condition.
 */

initial_run :-
    Iter is 0,
    asserta(min_test(0)),
    asserta(min_variable(1,0)),

    /* Run the model at the initial point */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    asserta(result(Cummd)),
    output_cummd(Cummd,Iter).

/* Rule store_min_values stores the current values for the two variables
 * to the database for future retrieval.
 */
store_min_values :-
    abolish(min_test/1),
    abolish(min_variable/2),
    tabsize(TableSize),
    foriterate(save_min_table,TableSize),
    test(S_val),
    asserta(min_test(S_val)).

/* Rule save_min_table stores the current value for the table variable
 * to the corresponding value for min_variable for future retrieval.
 */
save_min_table :-
    counter(Iter),
    variable(Iter,Value),
    asserta(min_variable(Iter,Value)).


/* Rule get_cummd runs the model with the new value Val and then reads the
 * new value for CUMMD from the file project.out.
 */
get_cummd(Val,Iter,Cummd) :-
    /* Replace the old variable value in the database with the new value. */
    retract(variable(Iter,OldVal)),
```

```
    asserta(variable(Iter,Val)),

    /* Run the model and read the new value for CUMMD.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    output_cummd(Cummd,Iter).

/* Rule s_get_cummd runs the model with the new value val and then reads the
 * new value for CUMMD from the file project.out.
 */
s_get_cummd(Val,Iter,Cummd) :-
    /* Replace the old variable value in the database with the new value. */
    retract(test(OldVal)),
    asserta(test(Val)),

    /* Run the model and read the new value for CUMMD.  */
    output_values,
    shell(model),
    read_cummd,
    cummd(Cummd),
    output_cummd(Cummd,Iter).


/* Rule exit_condition checks to see if the pulse size for both variables is
 * less than the user specified minimums and if so writes the final value for
 * CUMMD to the file summary.dat and exits the program.
 */

exit_condition :-
    /* Retrieve the current and minimum pulse sizes from the database.  */
    [!t_size(TPulse),
    t_minsize(TMinPulse),
    s_size(SPulse),
    s_minsize(SMinPulse)!],

    TPulse < TMinPulse, SPulse < SMinPulse,

    /* Retrieve the values resulting in the minimum CUMMD and run the model at this
     * point so that the final values in the file project.rsl will reflect the
     * values at the minimum point.
     */
    output_message('The Final Result '),
    retrieve_min_values,
    output_values,
    shell(model),
    open(S,'summary.dat',a),
    result(MinCummd),
    write(S,'The final cummd is: '),
    write(S,MinCummd),nl(S),
    close(S),
```

```
    halt.


/* Rule read_cummd reads the new value of CUMMD from the file project.out
 * and stores it in the database after a simulation has been run.
 */
read_cummd :-
    open(C,'project.out',r),
    read(C,Output),
    abolish(cummd/1),
    asserta(Output),
    close(C).

/* Rule output_cummd writes the value of Cummd to summary.dat */
output_cummd(Cummd,Iter) :-
    open(S,'summary.dat',a),
    write(S,Iter),write(S,'. '),
    write(S,'CUMMD= '),
    write(S,Cummd),nl(S),
    close(S).

/* Rule output_values outputs the values of the variable to project.dnx for
 * use in the simulation and summary.dat for post-run analysis.
 */
 output_values :-
    t_var(TVarName),
    tabsize(TabSize),
    test(SVal),
    s_var(SVarName),

    /* Write the values to project.dnx  */
    create(D,'project.dnx'),
    write(D,'T '), write(D,TVarName), write(D,'='),
    foriterate(write_val1(D),TabSize),nl(D),
    write(D,'C '), write(D,SVarName), write(D,'='),
    write(D,SVal),nl(D),
    close(D),

    /* Write the values to summary.dat */
    open(S,'summary.dat',a),
    write(S,TVarName),write(S,'='),
    foriterate(write_val2(S),TabSize),nl(S),
    write(S,SVarName), write(S,'='), write(S,SVal),
    nl(S), close(S).

/* Rule write_val1 is used to write the a value in the format required in
 * the file project.dnx.
 */
write_val1(F) :-
    counter(I),
    variable(I,Val),
    write(F,Val), write(F,' ').
```

```
/* Rule write_val2 is used to write a value in the format required in the
 * file summary.dat.
 */
write_val2(F) :-
    counter(I),
    variable(I,Val),
    write(F,Val), write(F,'/').

/* Rule retrieve_min_values replaces the current variable values with
 * the minimum values stored in the database.
 */
retrieve_min_values :-
    abolish(test/1),
    abolish(variable/2),
    tabsize(TableSize),
    foriterate(retrieve_min_table,TableSize),
    min_test(S_val),
    asserta(test(S_val)).

/* Rule retrieve_min_table retrieve the minimum value for the table
 * variable to the corresponding value for variable.
 */
retrieve_min_table :-
    counter(Iter),
    min_variable(Iter,Value),
    asserta(variable(Iter,Value)).

/* Rule get_initial_values prompts the user for the initial values
 * for an entry in the table and then store that value in the
 * appropriate element of the predicate variable.
 */

get_initial_values :-
    counter(Iter),
    write('Enter Table Value '), write(Iter), write('. '),
    readnumber(Val),
    asserta(variable(Iter,Val)).

/* Rule output_header writes the user specified values to the file
 * summary.dat to aid in post-run analysis.
 */
output_header(TVarName,SVarName,TPulse,TMinPulse,SPulse,SMinPulse) :-
    create(S,'summary.dat'),
    write(S,'       Table Variable Name       = '),write(S,TVarName),nl(S),
    write(S,'       Single Variable Name      = '),write(S,SVarName),nl(S),
    write(S,'       Table Pulse size factor   = '),write(S,TPulse),nl(S),
    write(S,'       Table Minimum Pulse Size  = '),write(S,TMinPulse),nl(S),
    write(S,'       Single Pulse size factor  = '),write(S,SPulse),nl(S),
    write(S,'       Single Minimum Pulse Size = '),write(S,SMinPulse),nl(S),
    nl(S),
    close(S).
```

```
/********************** Utility Routines **********************************/

/* Rule foriterate performs iteration on the predicate Pred from 1 to N
 * similar to a for loop in Pascal or C.
 */
foriterate(Pred,N) :-
    asserta(counter(0)),
    repeat,
          [!counter(K),
           K2 is K + 1,
           retract(counter(K)),
           asserta(counter(K2)),
           iterate2(Pred)!],
    K2 >= N,
    retract(counter(K2)),!.

/* Rule iterate2 is used by foriterate to execute the predicate Pred. */

iterate2(Pred) :-
    (call(Pred),!), 1=1.

/*  Rule output_message writes the message passed to it to the
 *  file summary.dat.
 */
output_message(Message) :-
    open(S,'summary.dat',a),
    nl(S),write(S,'**** '),write(S,Message),write(S,' ****'),nl(S),
    close(S).

/* Rule range_check ensures that the input variable is between the
 * minimum and maximum value for that variable and if not assigns
 * the minimum or maximum value to the result.
 */

/* If the value is greater than the maximum then return the maximum value */
range_check(Val,Max,Min,Result) :-
    Val > Max,
    Result is Max,!.

/* If the value is less than the minimum then return the minimum value.  */
range_check(Val,Max,Min,Result) :-
    Val < Min,
    Result is Min,!.

/* Otherwise return the initial value Val.  */
range_check(Val,Max,Min,Result) :-
    Result is Val,!.

/* Rule readnumber reads in a value from the user as a string and then
 * converts it to a prolog number.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and float_text are built in Arity prolog predicates and are not
```

```
 * standard prolog.
 */

readnumber(Result) :-
    read_string(255,ArString),nl,
    float_text(Result,ArString,_).

readnumber(Result) :-
    write('Number expected, try again'),
    nl,
    readnumber(Result).

/* Rule readinteger reads in a value from the user as a string and then
 * converts it to a prolog integer.  It is used to avoid making the user
 * type a period after each number input. NOTE-- the predicates read_string
 * and int_text are built in Arity prolog predicates and are not
 * standard prolog.
 */

readinteger(Result) :-
    read_string(255,ArString),nl,
    int_text(Result,ArString).

readinteger(Result) :-
    write('Integer expected, try again'),
    nl,
    readnumber(Result).
```

APPENDIX E

Table Variable Name = TPFMQA
Single Variable Name = DEVPRT
Table Pulse size factor = 0.05
Table Minimum Pulse Size = 0.005
Single Pulse size factor = 0.05
Single Minimum Pulse Size = 0.005

TPFMQA=0.325/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
0. CUMMD= 2093.08
TPFMQA=0.375/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
1. CUMMD= 2121.87
TPFMQA=0.275/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
1. CUMMD= 2051.57
TPFMQA=0.275/0.34/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
2. CUMMD= 2123.12
TPFMQA=0.275/0.24/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
2. CUMMD= 1996.79
TPFMQA=0.275/0.24/0.325/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
3. CUMMD= 2043.12
TPFMQA=0.275/0.24/0.225/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
3. CUMMD= 1954.28
TPFMQA=0.275/0.24/0.225/0.305/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
4. CUMMD= 1987.52
TPFMQA=0.275/0.24/0.225/0.205/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
4. CUMMD= 1934.73
TPFMQA=0.275/0.24/0.225/0.205/0.3/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
5. CUMMD= 1954.31
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
5. CUMMD= 1902.49
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.325/0.325/0.375/0.4/0.4/
DEVPRT=0.85
6. CUMMD= 1921.01
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.325/0.375/0.4/0.4/
DEVPRT=0.85
6. CUMMD= 1885.17
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.375/0.375/0.4/0.4/

DEVPRT=0.85
7. CUMMD= 1901.03
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.375/0.4/0.4/
DEVPRT=0.85
7. CUMMD= 1869.75
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.425/0.4/0.4/
DEVPRT=0.85
8. CUMMD= 1884.09
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.4/0.4/
DEVPRT=0.85
8. CUMMD= 1855.91
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.45/0.4/
DEVPRT=0.85
9. CUMMD= 1883.77
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.4/
DEVPRT=0.85
9. CUMMD= 1840.59
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.45/
DEVPRT=0.85
10. CUMMD= 1882.48
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.85
10. CUMMD= 1813.24

Base CUMMD is 2093.08
Final Temp Head CUMMD is 1813.24

**** Jump Pattern ****
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
0. CUMMD= 1679.15
TPFMQA=0.275/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
1. CUMMD= 1687.31
TPFMQA=0.175/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
1. CUMMD= 1696.56
TPFMQA=0.225/0.24/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
2. CUMMD= 1698.27
TPFMQA=0.225/0.14/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
2. CUMMD= 1671.6
TPFMQA=0.225/0.14/0.225/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
3. CUMMD= 1687.05
TPFMQA=0.225/0.14/0.125/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
3. CUMMD= 1648.58
TPFMQA=0.225/0.14/0.125/0.205/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
4. CUMMD= 1665.99

TPFMQA=0.225/0.14/0.125/0.105/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
4. CUMMD= 1633.96
TPFMQA=0.225/0.14/0.125/0.105/0.2/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
5. CUMMD= 1646.29
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
5. CUMMD= 1615.7
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.225/0.225/0.275/0.3/0.3/
DEVPRT=0.85
6. CUMMD= 1628.8
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.225/0.275/0.3/0.3/
DEVPRT=0.85
6. CUMMD= 1609.53
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.275/0.275/0.3/0.3/
DEVPRT=0.85
7. CUMMD= 1616.24
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.275/0.3/0.3/
DEVPRT=0.85
7. CUMMD= 1602.9
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.325/0.3/0.3/
DEVPRT=0.85
8. CUMMD= 1609.82
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.3/0.3/
DEVPRT=0.85
8. CUMMD= 1596.07
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.35/0.3/
DEVPRT=0.85
9. CUMMD= 1602.89
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.3/
DEVPRT=0.85
9. CUMMD= 1589.24
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.35/
DEVPRT=0.85
10. CUMMD= 1596.07
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
10. CUMMD= 1589.08

Base CUMMD is 1813.24
Final Temp Head CUMMD is 1589.08

**** Jump Pattern ****
TPFMQA=0.175/0.04/0.025/0.01/0.01/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85
0. CUMMD= 1672.19
TPFMQA=0.225/0.04/0.025/0.01/0.01/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85
1. CUMMD= 1646.27
TPFMQA=0.225/0.09/0.025/0.01/0.01/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85

2. CUMMD= 1619.73
TPFMQA=0.225/0.09/0.075/0.01/0.01/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85
3. CUMMD= 1611.61
TPFMQA=0.225/0.09/0.075/0.06/0.01/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85
4. CUMMD= 1603.37
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.025/0.075/0.125/0.15/0.15/
DEVPRT=0.85
5. CUMMD= 1599.38
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.075/0.125/0.15/0.15/
DEVPRT=0.85
6. CUMMD= 1590.47
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.125/0.15/0.15/
DEVPRT=0.85
7. CUMMD= 1589.56
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.175/0.15/0.15/
DEVPRT=0.85
8. CUMMD= 1589.45
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.175/0.2/0.15/
DEVPRT=0.85
9. CUMMD= 1589.45
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.175/0.1/0.15/
DEVPRT=0.85
9. CUMMD= 1589.45
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.175/0.15/0.2/
DEVPRT=0.85
10. CUMMD= 1589.45
TPFMQA=0.225/0.09/0.075/0.06/0.06/0.075/0.125/0.175/0.15/0.1/
DEVPRT=0.85
10. CUMMD= 1589.45

Base CUMMD is 1589.08
Final Temp Head CUMMD is 1589.45

**** Retreat to Base for Local Exploration ****
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
0. CUMMD= 1589.08
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
1. CUMMD= 1587.02
TPFMQA=0.275/0.19/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
2. CUMMD= 1592.88
TPFMQA=0.275/0.09/0.125/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
2. CUMMD= 1591.53
TPFMQA=0.275/0.14/0.175/0.105/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
3. CUMMD= 1597.52
TPFMQA=0.275/0.14/0.075/0.105/0.1/0.125/0.175/0.225/0.25/0.25/

DEVPRT=0.85
3. CUMMD= 1582.44
TPFMQA=0.275/0.14/0.075/0.155/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
4. CUMMD= 1588.35
TPFMQA=0.275/0.14/0.075/0.055/0.1/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
4. CUMMD= 1569.43
TPFMQA=0.275/0.14/0.075/0.055/0.15/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
5. CUMMD= 1579.54
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
5. CUMMD= 1565.99
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.175/0.175/0.225/0.25/0.25/
DEVPRT=0.85
6. CUMMD= 1571.43
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.175/0.225/0.25/0.25/
DEVPRT=0.85
6. CUMMD= 1560.73
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.225/0.25/0.25/
DEVPRT=0.85
7. CUMMD= 1560.27
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.275/0.25/0.25/
DEVPRT=0.85
8. CUMMD= 1567.04
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.175/0.25/0.25/
DEVPRT=0.85
8. CUMMD= 1560.28
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.225/0.3/0.25/
DEVPRT=0.85
9. CUMMD= 1567.08
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.225/0.2/0.25/
DEVPRT=0.85
9. CUMMD= 1560.21
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.225/0.2/0.3/
DEVPRT=0.85
10. CUMMD= 1560.28
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.225/0.2/0.2/
DEVPRT=0.85
10. CUMMD= 1560.22

Base CUMMD is 1589.08
Final Temp Head CUMMD is 1560.21

**** Jump Patttern ****
TPFMQA=0.325/0.14/0.025/0.01/0.01/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
0. CUMMD= 1575.58
TPFMQA=0.375/0.14/0.025/0.01/0.01/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
1. CUMMD= 1565.12

TPFMQA=0.375/0.19/0.025/0.01/0.01/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
2. CUMMD= 1548.97
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
3. CUMMD= 1547.57
TPFMQA=0.375/0.19/0.075/0.06/0.01/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
4. CUMMD= 1554.31
TPFMQA=0.375/0.19/0.075/0.01/0.06/0.025/0.275/0.225/0.15/0.25/
DEVPRT=0.85
5. CUMMD= 1558.12
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.075/0.275/0.225/0.15/0.25/
DEVPRT=0.85
6. CUMMD= 1553.17
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.01/0.275/0.225/0.15/0.25/
DEVPRT=0.85
6. CUMMD= 1547.89
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.325/0.225/0.15/0.25/
DEVPRT=0.85
7. CUMMD= 1553.98
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.225/0.15/0.25/
DEVPRT=0.85
7. CUMMD= 1541.2
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.275/0.15/0.25/
DEVPRT=0.85
8. CUMMD= 1547.91
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.175/0.15/0.25/
DEVPRT=0.85
8. CUMMD= 1534.54
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.175/0.2/0.25/
DEVPRT=0.85
9. CUMMD= 1541.28
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.175/0.1/0.25/
DEVPRT=0.85
9. CUMMD= 1534.54
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.175/0.15/0.3/
DEVPRT=0.85
10. CUMMD= 1541.3
TPFMQA=0.375/0.19/0.075/0.01/0.01/0.025/0.225/0.175/0.15/0.2/
DEVPRT=0.85
10. CUMMD= 1534.53

Base CUMMD is 1560.21
Final Temp Head CUMMD is 1534.53

**** Jump Pattern ****
TPFMQA=0.475/0.24/0.075/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
0. CUMMD= 1528.6
TPFMQA=0.525/0.24/0.075/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85

1. CUMMD= 1539.36
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
1. CUMMD= 1522.61
TPFMQA=0.425/0.29/0.075/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
2. CUMMD= 1535.13
TPFMQA=0.425/0.19/0.075/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
2. CUMMD= 1529.97
TPFMQA=0.425/0.24/0.125/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
3. CUMMD= 1527.61
TPFMQA=0.425/0.24/0.025/0.01/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
3. CUMMD= 1523.61
TPFMQA=0.425/0.24/0.075/0.06/0.01/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
4. CUMMD= 1529.29
TPFMQA=0.425/0.24/0.075/0.01/0.06/0.01/0.225/0.125/0.1/0.15/
DEVPRT=0.85
5. CUMMD= 1526.54
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.06/0.225/0.125/0.1/0.15/
DEVPRT=0.85
6. CUMMD= 1528.27
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.275/0.125/0.1/0.15/
DEVPRT=0.85
7. CUMMD= 1529.04
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.125/0.1/0.15/
DEVPRT=0.85
7. CUMMD= 1516.28
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.175/0.1/0.15/
DEVPRT=0.85
8. CUMMD= 1522.86
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.075/0.1/0.15/
DEVPRT=0.85
8. CUMMD= 1516.35
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.125/0.15/0.15/
DEVPRT=0.85
9. CUMMD= 1516.33
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.125/0.05/0.15/
DEVPRT=0.85
9. CUMMD= 1516.29
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.125/0.1/0.2/
DEVPRT=0.85
10. CUMMD= 1522.93
TPFMQA=0.425/0.24/0.075/0.01/0.01/0.01/0.175/0.125/0.1/0.1/
DEVPRT=0.85
10. CUMMD= 1516.28

Base CUMMD is 1534.53
Final Temp Head CUMMD is 1516.28

**** Jump Pattern ****
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
0. CUMMD= 1500.89
TPFMQA=0.525/0.29/0.075/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
1. CUMMD= 1517.11
TPFMQA=0.425/0.29/0.075/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
1. CUMMD= 1502.47
TPFMQA=0.475/0.34/0.075/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
2. CUMMD= 1517.93
TPFMQA=0.475/0.24/0.075/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
2. CUMMD= 1509.16
TPFMQA=0.475/0.29/0.125/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
3. CUMMD= 1513.24
TPFMQA=0.475/0.29/0.025/0.01/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
3. CUMMD= 1501.4
TPFMQA=0.475/0.29/0.075/0.06/0.01/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
4. CUMMD= 1514.19
TPFMQA=0.475/0.29/0.075/0.01/0.06/0.01/0.125/0.075/0.05/0.1/
DEVPRT=0.85
5. CUMMD= 1511.39
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.06/0.125/0.075/0.05/0.1/
DEVPRT=0.85
6. CUMMD= 1513.01
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.175/0.075/0.05/0.1/
DEVPRT=0.85
7. CUMMD= 1513.72
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.075/0.05/0.1/
DEVPRT=0.85
7. CUMMD= 1494.66
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.125/0.05/0.1/
DEVPRT=0.85
8. CUMMD= 1501.16
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.025/0.05/0.1/
DEVPRT=0.85
8. CUMMD= 1488.24
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.025/0.1/0.1/
DEVPRT=0.85
9. CUMMD= 1494.71
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.025/0.01/0.1/
DEVPRT=0.85
9. CUMMD= 1494.67
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.025/0.05/0.15/
DEVPRT=0.85

10. CUMMD= 1494.75
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.075/0.025/0.05/0.05/
DEVPRT=0.85
10. CUMMD= 1494.66

Base CUMMD is 1516.28
Final Temp Head CUMMD is 1488.24

**** Jump Pattern ****
TPFMQA=0.525/0.34/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
0. CUMMD= 1487.74
TPFMQA=0.575/0.34/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
1. CUMMD= 1500.9
TPFMQA=0.475/0.34/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
1. CUMMD= 1479.26
TPFMQA=0.475/0.39/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
2. CUMMD= 1487.57
TPFMQA=0.475/0.29/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
2. CUMMD= 1495.21
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
3. CUMMD= 1479.16
TPFMQA=0.475/0.34/0.125/0.06/0.01/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
4. CUMMD= 1492.54
TPFMQA=0.475/0.34/0.125/0.01/0.06/0.01/0.01/0.01/0.01/0.05/
DEVPRT=0.85
5. CUMMD= 1489.8
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.06/0.01/0.01/0.01/0.05/
DEVPRT=0.85
6. CUMMD= 1491.19
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.06/0.01/0.01/0.05/
DEVPRT=0.85
7. CUMMD= 1491.76
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.06/0.01/0.05/
DEVPRT=0.85
8. CUMMD= 1485.59
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.06/0.05/
DEVPRT=0.85
9. CUMMD= 1485.61
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.1/
DEVPRT=0.85
10. CUMMD= 1485.68
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
10. CUMMD= 1472.75

Base CUMMD is 1488.24
Final Temp Head CUMMD is 1472.75

**** Jump Pattern ****
TPFMQA=0.475/0.39/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1509.98
TPFMQA=0.525/0.39/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1528.63
TPFMQA=0.425/0.39/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1492.07
TPFMQA=0.425/0.44/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1514.77
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1475.6
TPFMQA=0.425/0.34/0.225/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1484.42
TPFMQA=0.425/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1481.37
TPFMQA=0.425/0.34/0.175/0.06/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1488.86
TPFMQA=0.425/0.34/0.175/0.01/0.06/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1486.26
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.06/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1481.25
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.01/0.06/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1481.78
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.01/0.01/0.06/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1481.95
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.01/0.01/0.01/0.06/0.01/
DEVPRT=0.85
9. CUMMD= 1481.96
TPFMQA=0.425/0.34/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.06/
DEVPRT=0.85
10. CUMMD= 1482.04

Base CUMMD is 1472.75
Final Temp Head CUMMD is 1475.6

**** Retreat to Base for Local Exploration ****
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/

DEVPRT=0.85
0. CUMMD= 1472.75
TPFMQA=0.525/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1488.01
TPFMQA=0.425/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1481.37
TPFMQA=0.475/0.39/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1494.84
TPFMQA=0.475/0.29/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1481.41
TPFMQA=0.475/0.34/0.175/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1486.45
TPFMQA=0.475/0.34/0.075/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1479.25
TPFMQA=0.475/0.34/0.125/0.06/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1486.0
TPFMQA=0.475/0.34/0.125/0.01/0.06/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1483.29
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.06/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1484.65
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.06/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1478.91
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.06/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1479.09
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.06/0.01/
DEVPRT=0.85
9. CUMMD= 1479.12
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.06/
DEVPRT=0.85
10. CUMMD= 1479.2

Base CUMMD is 1472.75
Final Temp Head CUMMD is 1472.75

**** Reduce Table Pulse Size ****
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1472.75
TPFMQA=0.5/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1483.7

TPFMQA=0.45/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1474.11
TPFMQA=0.475/0.365/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1480.61
TPFMQA=0.475/0.315/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1477.24
TPFMQA=0.475/0.34/0.15/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1479.3
TPFMQA=0.475/0.34/0.1/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1472.79
TPFMQA=0.475/0.34/0.125/0.035/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1476.24
TPFMQA=0.475/0.34/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1477.99
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.035/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1478.66
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.035/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1478.94
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.035/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1479.04
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.035/0.01/
DEVPRT=0.85
9. CUMMD= 1479.05
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.035/
DEVPRT=0.85
10. CUMMD= 1479.09

Base CUMMD is 1472.75
Final Temp Head CUMMD is 1472.75

**** Reduce Table Pulse Size ****
TPFMQA=0.475/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1472.75
TPFMQA=0.4875/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1475.11
TPFMQA=0.4625/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1470.33
TPFMQA=0.4625/0.3525/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85

2. CUMMD= 1471.15
TPFMQA=0.4625/0.3275/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1475.74
TPFMQA=0.4625/0.34/0.1375/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1473.48
TPFMQA=0.4625/0.34/0.1125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1473.5
TPFMQA=0.4625/0.34/0.125/0.0225/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1475.14
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1469.79
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.0225/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1475.84
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.0225/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1475.98
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.0225/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1476.03
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.0225/0.01/
DEVPRT=0.85
9. CUMMD= 1476.04
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.0225/
DEVPRT=0.85
10. CUMMD= 1476.06

Base CUMMD is 1472.75
Final Temp Head CUMMD is 1469.79

**** Jump Pattern ****
TPFMQA=0.45/0.34/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1472.98
TPFMQA=0.4625/0.34/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1475.5
TPFMQA=0.4375/0.34/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1476.64
TPFMQA=0.45/0.3525/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1473.85
TPFMQA=0.45/0.3275/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1478.33
TPFMQA=0.45/0.34/0.1375/0.01/0.035/0.01/0.01/0.01/0.01/0.01/

DEVPRT=0.85
3. CUMMD= 1476.14
TPFMQA=0.45/0.34/0.1125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1476.13
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1471.57
TPFMQA=0.45/0.34/0.125/0.0225/0.0475/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1477.28
TPFMQA=0.45/0.34/0.125/0.0225/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1472.11
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.0225/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1477.63
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.01/0.0225/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1477.77
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.01/0.01/0.0225/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1477.81
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.01/0.01/0.01/0.0225/0.01/
DEVPRT=0.85
9. CUMMD= 1477.82
TPFMQA=0.45/0.34/0.125/0.0225/0.035/0.01/0.01/0.01/0.01/0.0225/
DEVPRT=0.85
10. CUMMD= 1477.84

Base CUMMD is 1469.79
Final Temp Head CUMMD is 1471.57

**** Retreat to Base for Local Exploration ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1469.79
TPFMQA=0.475/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1472.24
TPFMQA=0.45/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1473.55
TPFMQA=0.4625/0.3525/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1476.9
TPFMQA=0.4625/0.3275/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1475.17
TPFMQA=0.4625/0.34/0.1375/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1472.99

TPFMQA=0.4625/0.34/0.1125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1472.93
TPFMQA=0.4625/0.34/0.125/0.0225/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1474.64
TPFMQA=0.4625/0.34/0.125/0.01/0.035/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1475.5
TPFMQA=0.4625/0.34/0.125/0.01/0.01/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1470.33
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.0225/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1475.84
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.0225/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1475.98
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.0225/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1476.03
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.0225/0.01/
DEVPRT=0.85
9. CUMMD= 1476.04
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.0225/
DEVPRT=0.85
10. CUMMD= 1476.06

Base CUMMD is 1469.79
Final Temp Head CUMMD is 1469.79

**** Reduce Table Pulse Size ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1469.79
TPFMQA=0.46875/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1474.14
TPFMQA=0.45625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
1. CUMMD= 1471.68
TPFMQA=0.4625/0.34625/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1473.35
TPFMQA=0.4625/0.33375/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
2. CUMMD= 1472.49
TPFMQA=0.4625/0.34/0.13125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
3. CUMMD= 1474.49
TPFMQA=0.4625/0.34/0.11875/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85

3. CUMMD= 1471.35
TPFMQA=0.4625/0.34/0.125/0.01625/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
4. CUMMD= 1475.33
TPFMQA=0.4625/0.34/0.125/0.01/0.02875/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1475.76
TPFMQA=0.4625/0.34/0.125/0.01/0.01625/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
5. CUMMD= 1470.06
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01625/0.01/0.01/0.01/0.01/
DEVPRT=0.85
6. CUMMD= 1475.93
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01625/0.01/0.01/0.01/
DEVPRT=0.85
7. CUMMD= 1476.0
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01625/0.01/0.01/
DEVPRT=0.85
8. CUMMD= 1476.03
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01625/0.01/
DEVPRT=0.85
9. CUMMD= 1476.03
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01625/
DEVPRT=0.85
10. CUMMD= 1476.04

Base CUMMD is 1469.79
Final Temp Head CUMMD is 1469.79

**** Reduce Table Pulse Size ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1469.79
The min cummd after optimizing table function is: 1469.79
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1469.79
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.8
1. CUMMD= 1484.45
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.9
1. CUMMD= 1509.23

**** Reduce single variable pulse size ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.825
1. CUMMD= 1470.13
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.875
1. CUMMD= 1489.04

**** Reduce single variable pulse size ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.8375
1. CUMMD= 1473.0
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.8625
1. CUMMD= 1479.31

**** Reduce single variable pulse size ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.84375
1. CUMMD= 1474.49
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85625
1. CUMMD= 1477.64

**** Reduce single variable pulse size ****

**** The final values are ****
TPFMQA=0.4625/0.34/0.125/0.01/0.0225/0.01/0.01/0.01/0.01/0.01/
DEVPRT=0.85
0. CUMMD= 1469.79

APPENDIX F

Table Variable Name = TPFMQA
Single Variable Name = DEVPRT
Table Pulse size factor = 0.05
Table Minimum Pulse Size = 0.005
Single Pulse size factor = 0.05
Single Minimum Pulse Size = 0.005

TPFMQA=0.325/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
0. CUMMD= 2093.08
TPFMQA=0.375/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
1. CUMMD= 2121.87
TPFMQA=0.275/0.29/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
1. CUMMD= 2051.57
TPFMQA=0.275/0.34/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
2. CUMMD= 2123.12
TPFMQA=0.275/0.24/0.275/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
2. CUMMD= 1996.79
TPFMQA=0.275/0.24/0.325/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
3. CUMMD= 2043.12
TPFMQA=0.275/0.24/0.225/0.255/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
3. CUMMD= 1954.28
TPFMQA=0.275/0.24/0.225/0.305/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
4. CUMMD= 1987.52
TPFMQA=0.275/0.24/0.225/0.205/0.25/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
4. CUMMD= 1934.73
TPFMQA=0.275/0.24/0.225/0.205/0.3/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
5. CUMMD= 1954.31
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.275/0.325/0.375/0.4/0.4/
DEVPRT=0.85
5. CUMMD= 1902.49
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.325/0.325/0.375/0.4/0.4/
DEVPRT=0.85
6. CUMMD= 1921.01
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.325/0.375/0.4/0.4/
DEVPRT=0.85
6. CUMMD= 1885.17
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.375/0.375/0.4/0.4/

DEVPRT=0.85
7. CUMMD= 1901.03
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.375/0.4/0.4/
DEVPRT=0.85
7. CUMMD= 1869.75
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.425/0.4/0.4/
DEVPRT=0.85
8. CUMMD= 1884.09
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.4/0.4/
DEVPRT=0.85
8. CUMMD= 1855.91
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.45/0.4/
DEVPRT=0.85
9. CUMMD= 1883.77
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.4/
DEVPRT=0.85
9. CUMMD= 1840.59
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.45/
DEVPRT=0.85
10. CUMMD= 1882.48
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.85
10. CUMMD= 1813.24
TPFMQA=0.275/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
1. CUMMD= 1801.32
TPFMQA=0.325/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
1. CUMMD= 1817.43
TPFMQA=0.225/0.24/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
1. CUMMD= 1787.37
TPFMQA=0.225/0.29/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
2. CUMMD= 1818.89
TPFMQA=0.225/0.19/0.225/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
2. CUMMD= 1769.28
TPFMQA=0.225/0.19/0.275/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
3. CUMMD= 1794.46
TPFMQA=0.225/0.19/0.175/0.205/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
3. CUMMD= 1746.31
TPFMQA=0.225/0.19/0.175/0.255/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
4. CUMMD= 1763.83
TPFMQA=0.225/0.19/0.175/0.155/0.2/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
4. CUMMD= 1730.13
TPFMQA=0.225/0.19/0.175/0.155/0.25/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9

5. CUMMD= 1739.28
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.225/0.275/0.325/0.35/0.35/
DEVPRT=0.9
5. CUMMD= 1712.68
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.275/0.275/0.325/0.35/0.35/
DEVPRT=0.9
6. CUMMD= 1721.83
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.275/0.325/0.35/0.35/
DEVPRT=0.9
6. CUMMD= 1703.57
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.325/0.325/0.35/0.35/
DEVPRT=0.9
7. CUMMD= 1712.77
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.325/0.35/0.35/
DEVPRT=0.9
7. CUMMD= 1685.76
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.375/0.35/0.35/
DEVPRT=0.9
8. CUMMD= 1694.9
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.35/0.35/
DEVPRT=0.9
8. CUMMD= 1677.16
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.4/0.35/
DEVPRT=0.9
9. CUMMD= 1685.74
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.35/
DEVPRT=0.9
9. CUMMD= 1677.0
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.4/
DEVPRT=0.9
10. CUMMD= 1685.85
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
10. CUMMD= 1668.45
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.95
1. CUMMD= 1690.34
TPFMQA=0.225/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.85
2. CUMMD= 1679.15
TPFMQA=0.275/0.19/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
1. CUMMD= 1667.53
TPFMQA=0.275/0.24/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
2. CUMMD= 1685.29
TPFMQA=0.275/0.14/0.175/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
2. CUMMD= 1662.0
TPFMQA=0.275/0.14/0.225/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
3. CUMMD= 1676.09

TPFMQA=0.275/0.14/0.125/0.155/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
3. CUMMD= 1647.75
TPFMQA=0.275/0.14/0.125/0.205/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
4. CUMMD= 1664.9
TPFMQA=0.275/0.14/0.125/0.105/0.15/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
4. CUMMD= 1641.53
TPFMQA=0.275/0.14/0.125/0.105/0.2/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
5. CUMMD= 1646.31
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.275/0.3/0.3/
DEVPRT=0.9
5. CUMMD= 1623.12
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.225/0.225/0.275/0.3/0.3/
DEVPRT=0.9
6. CUMMD= 1636.52
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.125/0.225/0.275/0.3/0.3/
DEVPRT=0.9
6. CUMMD= 1624.18
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.275/0.275/0.3/0.3/
DEVPRT=0.9
7. CUMMD= 1637.23
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.175/0.275/0.3/0.3/
DEVPRT=0.9
7. CUMMD= 1623.33
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.325/0.3/0.3/
DEVPRT=0.9
8. CUMMD= 1637.51
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.3/0.3/
DEVPRT=0.9
8. CUMMD= 1623.06
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.35/0.3/
DEVPRT=0.9
9. CUMMD= 1623.1
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.3/
DEVPRT=0.9
9. CUMMD= 1615.96
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.35/
DEVPRT=0.9
10. CUMMD= 1623.04
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.9
10. CUMMD= 1608.95
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.95
1. CUMMD= 1634.49
TPFMQA=0.275/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
2. CUMMD= 1599.83
TPFMQA=0.325/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/

DEVPRT=0.85
1. CUMMD= 1604.4
TPFMQA=0.225/0.14/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
1. CUMMD= 1601.7
TPFMQA=0.275/0.19/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
2. CUMMD= 1613.12
TPFMQA=0.275/0.09/0.125/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
2. CUMMD= 1603.85
TPFMQA=0.275/0.14/0.175/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
3. CUMMD= 1617.8
TPFMQA=0.275/0.14/0.075/0.105/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
3. CUMMD= 1594.83
TPFMQA=0.275/0.14/0.075/0.155/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
4. CUMMD= 1607.71
TPFMQA=0.275/0.14/0.075/0.055/0.1/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
4. CUMMD= 1581.52
TPFMQA=0.275/0.14/0.075/0.055/0.15/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
5. CUMMD= 1592.07
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.175/0.225/0.225/0.25/0.25/
DEVPRT=0.85
5. CUMMD= 1577.86
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.225/0.225/0.225/0.25/0.25/
DEVPRT=0.85
6. CUMMD= 1583.52
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.225/0.25/0.25/
DEVPRT=0.85
6. CUMMD= 1565.62
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.275/0.225/0.25/0.25/
DEVPRT=0.85
7. CUMMD= 1578.76
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.175/0.225/0.25/0.25/
DEVPRT=0.85
7. CUMMD= 1565.99
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.275/0.25/0.25/
DEVPRT=0.85
8. CUMMD= 1579.17
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.25/0.25/
DEVPRT=0.85
8. CUMMD= 1565.57
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.3/0.25/
DEVPRT=0.85
9. CUMMD= 1565.65
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.85

9. CUMMD= 1558.79
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.3/
DEVPRT=0.85
10. CUMMD= 1565.58
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.2/
DEVPRT=0.85
10. CUMMD= 1558.79
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.9
1. CUMMD= 1598.08
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1548.1
TPFMQA=0.325/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1552.87
TPFMQA=0.225/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1557.82
TPFMQA=0.275/0.19/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1554.19
TPFMQA=0.275/0.09/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1565.63
TPFMQA=0.275/0.14/0.125/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1560.64
TPFMQA=0.275/0.14/0.025/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1555.34
TPFMQA=0.275/0.14/0.075/0.105/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1562.52
TPFMQA=0.275/0.14/0.075/0.01/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1554.49
TPFMQA=0.275/0.14/0.075/0.055/0.1/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1559.16
TPFMQA=0.275/0.14/0.075/0.055/0.01/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1551.22
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.175/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1561.28
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.075/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1550.23
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.275/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1554.69

TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.175/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1549.45
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.225/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1555.03
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.125/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1549.21
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.25/0.25/
DEVPRT=0.8
9. CUMMD= 1555.58
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.15/0.25/
DEVPRT=0.8
9. CUMMD= 1549.13
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.3/
DEVPRT=0.8
10. CUMMD= 1555.53
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.2/
DEVPRT=0.8
10. CUMMD= 1549.12
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.85
1. CUMMD= 1558.79
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.75
2. CUMMD= 1559.43
TPFMQA=0.3/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1550.46
TPFMQA=0.25/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1553.52
TPFMQA=0.275/0.165/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1550.49
TPFMQA=0.275/0.115/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1554.12
TPFMQA=0.275/0.14/0.1/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1553.85
TPFMQA=0.275/0.14/0.05/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1555.81
TPFMQA=0.275/0.14/0.075/0.08/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1551.8
TPFMQA=0.275/0.14/0.075/0.03/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1552.17
TPFMQA=0.275/0.14/0.075/0.055/0.075/0.125/0.225/0.175/0.2/0.25/

DEVPRT=0.8
5. CUMMD= 1553.23
TPFMQA=0.275/0.14/0.075/0.055/0.025/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1550.45
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.15/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1554.98
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.1/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1549.67
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.25/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1548.43
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.2/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1549.29
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.2/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1548.5
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.15/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1549.17
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.225/0.25/
DEVPRT=0.8
9. CUMMD= 1548.48
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.175/0.25/
DEVPRT=0.8
9. CUMMD= 1549.13
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.275/
DEVPRT=0.8
10. CUMMD= 1548.78
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.225/
DEVPRT=0.8
10. CUMMD= 1549.12
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.825
1. CUMMD= 1556.89
TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.775
2. CUMMD= 1553.55
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1545.97
TPFMQA=0.2875/0.1525/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1546.74
TPFMQA=0.2875/0.1275/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1552.69
TPFMQA=0.2875/0.14/0.0875/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8

3. CUMMD= 1548.7
TPFMQA=0.2875/0.14/0.0625/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1550.31
TPFMQA=0.2875/0.14/0.075/0.0675/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1551.18
TPFMQA=0.2875/0.14/0.075/0.0425/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1548.41
TPFMQA=0.2875/0.14/0.075/0.055/0.0625/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1552.39
TPFMQA=0.2875/0.14/0.075/0.055/0.0375/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1547.52
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.1375/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1552.24
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.1125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1546.99
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.2375/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1552.94
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.2125/0.175/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1546.18
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1545.88
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2125/0.25/
DEVPRT=0.8
9. CUMMD= 1552.75
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.1875/0.25/
DEVPRT=0.8
9. CUMMD= 1546.31
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.2625/
DEVPRT=0.8
10. CUMMD= 1553.0
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.2375/
DEVPRT=0.8
10. CUMMD= 1546.2
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8125
1. CUMMD= 1550.1
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.7875
2. CUMMD= 1553.65
TPFMQA=0.3/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1550.53

TPFMQA=0.275/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1548.71
TPFMQA=0.2875/0.1525/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1553.46
TPFMQA=0.2875/0.1275/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1552.65
TPFMQA=0.2875/0.14/0.0875/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1556.23
TPFMQA=0.2875/0.14/0.0625/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1550.27
TPFMQA=0.2875/0.14/0.075/0.0675/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1550.8
TPFMQA=0.2875/0.14/0.075/0.0425/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1547.74
TPFMQA=0.2875/0.14/0.075/0.055/0.0625/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1552.42
TPFMQA=0.2875/0.14/0.075/0.055/0.0375/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1546.27
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.1375/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1552.37
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.1125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1546.44
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.2375/0.1875/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1553.03
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.2125/0.1875/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1546.47
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.2/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1552.37
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1545.97
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2125/0.25/
DEVPRT=0.8
9. CUMMD= 1552.75
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.1875/0.25/
DEVPRT=0.8
9. CUMMD= 1546.31
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.2625/

DEVPRT=0.8
10. CUMMD= 1553.0
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.2375/
DEVPRT=0.8
10. CUMMD= 1546.2
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8125
1. CUMMD= 1550.1
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.7875
2. CUMMD= 1553.65
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1551.43
TPFMQA=0.28125/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
1. CUMMD= 1547.14
TPFMQA=0.2875/0.14625/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1549.95
TPFMQA=0.2875/0.13375/0.075/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
2. CUMMD= 1549.07
TPFMQA=0.2875/0.14/0.08125/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1550.88
TPFMQA=0.2875/0.14/0.06875/0.055/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
3. CUMMD= 1547.5
TPFMQA=0.2875/0.14/0.075/0.06125/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1552.29
TPFMQA=0.2875/0.14/0.075/0.04875/0.05/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
4. CUMMD= 1547.0
TPFMQA=0.2875/0.14/0.075/0.055/0.05625/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1552.88
TPFMQA=0.2875/0.14/0.075/0.055/0.04375/0.125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
5. CUMMD= 1546.38
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.13125/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1552.32
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.11875/0.225/0.1875/0.2/0.25/
DEVPRT=0.8
6. CUMMD= 1545.97
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.23125/0.1875/0.2/0.25/
DEVPRT=0.8
7. CUMMD= 1552.67
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.21875/0.1875/0.2/0.25/
DEVPRT=0.8

7. CUMMD= 1546.44
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.19375/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1552.83
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.2/0.25/
DEVPRT=0.8
8. CUMMD= 1546.26
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.20625/0.25/
DEVPRT=0.8
9. CUMMD= 1553.01
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
9. CUMMD= 1545.79
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25625/
DEVPRT=0.8
10. CUMMD= 1553.04
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.24375/
DEVPRT=0.8
10. CUMMD= 1545.97
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.80625
1. CUMMD= 1548.07
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.79375
2. CUMMD= 1550.15
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
1. CUMMD= 1551.71
TPFMQA=0.28125/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
1. CUMMD= 1547.16
TPFMQA=0.2875/0.14625/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1549.54
TPFMQA=0.2875/0.13375/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1548.43
TPFMQA=0.2875/0.14/0.08125/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
3. CUMMD= 1551.14
TPFMQA=0.2875/0.14/0.06875/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
3. CUMMD= 1547.91
TPFMQA=0.2875/0.14/0.075/0.06125/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1551.56
TPFMQA=0.2875/0.14/0.075/0.04875/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1546.43
TPFMQA=0.2875/0.14/0.075/0.055/0.05625/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1552.18

TPFMQA=0.2875/0.14/0.075/0.055/0.04375/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1546.72
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.13125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1552.6
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.11875/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1546.33
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.23125/0.1875/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1552.95
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.21875/0.1875/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1545.92
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.19375/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1553.1
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1545.7
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.2/0.25/
DEVPRT=0.8
9. CUMMD= 1546.26
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.1875/0.25/
DEVPRT=0.8
9. CUMMD= 1546.1
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25625/
DEVPRT=0.8
10. CUMMD= 1546.44
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.24375/
DEVPRT=0.8
10. CUMMD= 1545.93
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.80625
1. CUMMD= 1547.83
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.79375
2. CUMMD= 1550.45
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
1. CUMMD= 1544.77
TPFMQA=0.29375/0.14625/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1548.31
TPFMQA=0.29375/0.13375/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1548.08
TPFMQA=0.29375/0.14/0.08125/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
3. CUMMD= 1549.69
TPFMQA=0.29375/0.14/0.06875/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/

DEVPRT=0.8
3. CUMMD= 1546.66
TPFMQA=0.29375/0.14/0.075/0.06125/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1551.14
TPFMQA=0.29375/0.14/0.075/0.04875/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1545.09
TPFMQA=0.29375/0.14/0.075/0.055/0.05625/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1550.74
TPFMQA=0.29375/0.14/0.075/0.055/0.04375/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1545.28
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.13125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1551.17
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.11875/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1544.85
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.23125/0.18125/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1551.54
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.21875/0.18125/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1545.28
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1551.71
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1545.07
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.2/0.25/
DEVPRT=0.8
9. CUMMD= 1551.87
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.1875/0.25/
DEVPRT=0.8
9. CUMMD= 1545.02
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25625/
DEVPRT=0.8
10. CUMMD= 1551.99
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.24375/
DEVPRT=0.8
10. CUMMD= 1544.93
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.80625
1. CUMMD= 1547.42
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.79375
2. CUMMD= 1548.72
TPFMQA=0.3/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8

1. CUMMD= 1550.29
TPFMQA=0.2875/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
1. CUMMD= 1545.7
TPFMQA=0.29375/0.14625/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1548.31
TPFMQA=0.29375/0.13375/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
2. CUMMD= 1548.08
TPFMQA=0.29375/0.14/0.08125/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
3. CUMMD= 1549.69
TPFMQA=0.29375/0.14/0.06875/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
3. CUMMD= 1546.66
TPFMQA=0.29375/0.14/0.075/0.06125/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1551.14
TPFMQA=0.29375/0.14/0.075/0.04875/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
4. CUMMD= 1545.09
TPFMQA=0.29375/0.14/0.075/0.055/0.05625/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1550.74
TPFMQA=0.29375/0.14/0.075/0.055/0.04375/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
5. CUMMD= 1545.28
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.13125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1551.17
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.11875/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
6. CUMMD= 1544.85
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.23125/0.18125/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1551.54
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.21875/0.18125/0.19375/0.25/
DEVPRT=0.8
7. CUMMD= 1545.28
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.1875/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1551.71
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.175/0.19375/0.25/
DEVPRT=0.8
8. CUMMD= 1545.07
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.2/0.25/
DEVPRT=0.8
9. CUMMD= 1551.87
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.1875/0.25/
DEVPRT=0.8
9. CUMMD= 1545.02

TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25625/
DEVPRT=0.8
10. CUMMD= 1551.99
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.24375/
DEVPRT=0.8
10. CUMMD= 1544.93
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.80625
1. CUMMD= 1547.42
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.79375
2. CUMMD= 1548.72

**** The Final Result ****
TPFMQA=0.29375/0.14/0.075/0.055/0.05/0.125/0.225/0.18125/0.19375/0.25/
DEVPRT=0.8
The final cummd is: 1544.77

LIST OF REFERENCES

1. Abdel-Hamid, T. K., and Madnick, S. E., "Modeling the Dynamics of Software Project Management", CISR WP No. 163, MIT, Cambridge, MA, September 1987.

2. Abdel-Hamid, T. K., and Madnick, S. E., *Software Project Management*, pp. 148-231, Prentice-Hall, Inc., to be published in 1990.

3. Rowe, Neil C., *Artificial Intelligence Through Prolog*, pp. 191-207, Prentice-Hall, Inc., 1988.

4. Abdel-Hamid, T. K., and Leidy, F. H., "An Expert-Simulator for Allocating the Quality Assurance Effort in Software Development", pp. 4-5, Submitted for publication to *Simulation*, May 1989.

5. Leidy, F. H., *Design and Development of an Expert System Based Quality Assurance Module for the Dynamo Model of Software Project Management*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.

6. Wilde, Douglass J., *Optimum Seeking Methods*, pp. 145-150, Prentice-Hall Inc., 1964.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Tarek K. Abdel-Hamid, Code 54AH Department of Administrative Sciences Naval Postgraduate School Monterey, CA 93943-5000	5
4.	Dr. Tung Bui, Code 54BD Department of Administrative Sciences Naval Postgraduate School Monterey, CA 93943-5000	1
5.	CDR T. J. Hoskins Computer Technology (37) Naval Postgraduate School Monterey, CA 93943-5000	1
6.	LT Christopher E. Agan RD#2 Box 279A Johnsonville, NY 12094	2